Lecture Notes in Artificial Intelligence 1972
Subseries of Lecture Notes in Computer Science

Andrea Omicini Robert Tolksdorf
Franco Zambonelli (Eds.)

Engineering Societies in the Agents World

First International Workshop, ESAW 2000
Berlin, Germany, August 2000
Revised Papers

Springer

Lecture Notes in Artificial Intelligence 1972

Subseries of Lecture Notes in Computer Science
Edited by J. G. Carbonell and J. Siekmann

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Andrea Omicini Robert Tolksdorf
Franco Zambonelli (Eds.)

Engineering Societies in the Agents World

First International Workshop, ESAW 2000
Berlin, Germany, August 21, 2000
Revised Papers

Springer

Series Editors

Jaime G. Carbonell,Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Andrea Omicini
Università di Bologna
Dipartimento di Elettronica, Informatica e Sistemistica
Viale Risorgimento 2, 40136 Bologna, Italy
E-mail: aomicini@deis.unibo.it

Robert Tolksdorf
Technische Universität Berlin
Fachbereich 13 - Informatik
Franklinstr. 28/29, 10587 Berlin, Germany
E-mail: tolk@cs.tu-berlin.de

Franco Zambonelli
Università di Modena e Reggio Emilia
Dipartimento di Scienze dell'Ingegneria
Via Vignolese 905, 41100 Modena, Italy
E-mail: franco.zambonelli@unimo.it

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Engineering societies in the agents world : first international workshop ; revised papers / ESAW 2000, Berlin, Germany, August 21, 2000. Andrea Omicini ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1972 : Lecture notes in artificial intelligence)
ISBN 3-540-41477-0

CR Subject Classification (1998): I.2.11, I.2, D.2, C.2.4, H.4.3, H.5.3, J.4

ISBN 3-540-41477-0 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN 10781145 06/3142 5 4 3 2 1 0

Preface

The ESAW'00 workshop on Engineering Societies in the Agents' World was held on 21 August 2000 in conjunction with the 14th European Conference on Artificial Intelligence in Berlin at Humboldt University.

The recent research and technology advances in the area of Distributed Artificial Intelligence (DAI) are paving the way towards building new worlds. In the near future, multitudes of autonomous software agents are expected to be deployed in our networks and on the Web, to pursue goals on our behalf by communicating, synchronising, and either cooperating or competing with each other. These multi-agent systems (MAS) can no longer be conceived as static – multi-component – software architecture. Instead, due to the autonomous behaviour of agents and to the richness and dynamics of their interaction patterns, MAS can be better conceived in terms of *artificial societies* of individuals, living in and possibly roaming across a specific – often distributed – environment, and interacting according to patterns resembling those of human societies or complex eco-systems.

Given the above scenario, MAS are far beyond the boundaries of DAI, and researches in the area of MAS should necessarily gather contributions from many different and heterogeneous areas such as Distributed Systems, Social and Cognitive Sciences, Mobile Computing, and so on. In addition, although applications and systems built as societies of autonomous and intelligent agents promise to provide computer scientists and engineers with the expressive and computational power to tackle levels of complexity never reached before, specific abstraction, methodologies, and tools are required to enable an engineered approach to the construction of such systems. We feel an urgent need not only for theoretical foundations making MAS conceptual setting clear, but also for specific methodologies driving the design of agent societies, for specific technologies and processes driving their development, and for powerful and manageable infrastructures making agent societies a viable approach to embed intelligence into applications.

The above considerations motivated the organisation of the ESAW'00 workshop, devoted to discuss technologies, methodologies, and models for the engineering of complex applications based on societies of agents, and aimed at bringing together people and contributions from both within and outside the field of DAI by promoting cross-fertilisation. By focusing on the social aspects of MAS, ESAW'00 concentrated on the space of agent interaction, rather than on specific intra-agent issues, and on the technology and methodology issues rather than on the pure theoretical aspects.

In particular, the workshop aimed to address the following issues:

- coordination models and technologies for engineering agent societies
- analysis, design, development, and verification of agent societies
- engineering social intelligence and emergent behaviours in MAS

- application experiences in building agent societies
- centralised vs. decentralised social control
- interaction/coordination patterns in agent societies
- security and mobility issues in agent societies
- enabling infrastructures for agent societies
- methodologies, tools, and artifacts for engineering agent societies
- design vs. self-organisation

In response to the ESAW'00 call for papers, we received twenty papers, which were peer-reviewed for scope and quality. The ten best submitted papers were accepted for presentation and discussed at the workshop. After a further phase of review and expansion, also meant to incorporate the results of the workshop discussion, the selected papers were included in the present volume.

The quality of presentations at ESAW'00 was high, and triggered a highly interesting discussion amongst the 25 participants of the workshop – which we would sincerely like to thank for their active participation and the level of their contributions to the debate.

The central question of the discussion was what of the agent societies is to be modelled, and how should they be engineered. Around this very general question, several specific issues were discussed related to:

- engineering of emergent behaviours in agent societies
- models and roles of the environment
- modelling and engineering methodologies and standards

Concerning the behaviour of the society, emergence seemed to be a central issue, and all the participants agreed that a theory of emergence may be needed for the engineering of complex agent systems.

Opposed to common assumptions in software engineering, any equilibria of the agent system are explicitly not accepted, and changing roles is common. This behaviour seemed to be outside the scope of prescriptive modelling, and caused rather by some sort of "indirect programming". Engineering the self-evolution of an agent system and enabling adaptation, e.g., by selection, learning, negotiating, or intervention of the environment, was considered a central challenge. To do so, some structured agent-behavioural language was considered necessary in order to change parameters that affect the emergent behaviour. Platforms for the development of agent systems thus need to offer evaluation methods.

Since the study of emergent behaviours has to focus also on the environment in which agents live, other than on agents themselves and their interaction, the importance of identifying or modelling the agents' environment led to another major thread of discussion. Agent-based models should provide a view of MAS as something more than distributed systems, and also supply suitable abstractions to capture the environment in which they are situated.

In several cases, the environment in which a MAS is immersed should be considered as active. First, unforeseen interactions with the environment have to be expected, given that several real-world systems and applications exist in

dynamic and unpredictable environments. Second, open agent systems intrinsically deal with environments that entities can enter and leave at any time: this suggests that an active environment could be naturally exploited to model an open system. Finally, it may sometimes be necessary to embed specific interaction laws within the environment: this raises the issue of the models and enabling technologies allowing such laws to be represented and enforced.

Due to its active nature, the environment might be modelled as an agent, or as a set of agents. However, that may turn out to be the wrong abstraction level for several kinds of applications, and in particular for those that require an explicit modelling and engineering of agent-to-environment interactions, distinct from agent-to-agent interactions. There, it seems mandatory for an engineered approach to agent societies to adopt the environment itself as a primary abstraction in the design and development of agent societies.

Yet another issue discussed was that of modelling and design methodologies. While standard methodologies like UML could be adapted to model agent societies, it was unclear whether the current state of the UML technology is enough to capture all fundamental notions, and to express them at the most suitable level of abstraction. In this context, it was also stated that misusing abstractions as provided by standard methodologies can point to defects or a lack of expressiveness in these abstractions.

To integrate a different modelling perspective, one may consider layers of abstractions and use different models of roles and societies at various levels. Then, each of these levels could be engineered individually, as in the case, for instance, of reliable message passing. In that case, a lower level could be devoted to engineer a reliable message-exchange scheme between two agents, whereas an upper level could be devoted to engineer complex interaction protocols, based on the reliability provided by the lower level.

Finally, a brainstorm session collected requirements on modelling tools and infrastructures to support the insights from the discussion.

ESAW'00 was our first attempt to put researchers from different areas together to discuss the multi-faceted issues that emerge in the engineering of complex systems as societies of agents. Given the level of the contributions, we are confident that this volume will be useful to the agent community, by providing many original and heterogeneous views on such an interdisciplinary topic as well as several attempts to put everything together. It is our hope that ESAW'00 will be only the first event of a series, meant to provide the agent community with a forum where novel ideas and results can be shared by crossing the boundaries of the many research and application areas that meet in the agent field.

October 2000

Andrea Omicini
Robert Tolksdorf
Franco Zambonelli

Organisation

Organisers and Chairs

Andrea Omicini	*Università di Bologna, Italy*
Robert Tolksdorf	*Technische Universität Berlin, Germany*
Franco Zambonelli	*Università di Modena e Reggio Emilia, Italy*

Programme Committee

Cristiano Castelfranchi	*Università di Siena, Italy*
Paolo Ciancarini	*Università di Bologna, Italy*
Helder Coelho	*Universidade de Lisboa, Portugal*
Rino Falcone	*CNR, Italy*
Rune Gustavsson	*University of Karlskrona/Ronneby, Sweden*
Chihab Hanachi	*Université Toulouse 1, France*
Nick Jennings	*University of Southampton, UK*
Matthias Klusch	*DFKI, Germany*
Paolo Petta	*Austrian Research Institute for AI, Austria*
Agostino Poggi	*Università di Parma, Italy*
Antony Rowstron	*Microsoft Research, UK*
Christophe Sibertin-Blanc	*Université Toulouse 1, France*
Paul Tarau	*University of North Texas, USA*
Francesca Toni	*Imperial College, University of London, UK*

Additional Referees

Federico Bergenti	Fabrizio Riguzzi	Kostas Stathis
Giacomo Cabri	Giovanni Rimassa	Paolo Torroni
Marco Cremonini		

Table of Contents

Emerging Issues in Multiagent Systems Engineering

Engineering Social Order 1
Cristiano Castelfranchi

Distinguishing Environmental and Agent Dynamics:
A Case Study in Abstraction and Alternate Modeling Technologies 19
H. Van Dyke Parunak, Sven Brueckner, John Sauter and Robert S. Matthews

On Observing and Constraining Active Systems 34
Gianluca Moro and Mirko Viroli

Coordination Models and Technologies for Multiagent Systems

Context-Dependency in Internet-Agent Coordination 51
Giacomo Cabri, Letizia Leonardi and Franco Zambonelli

Coordination Issues in Multi-agent Event Data Processing 64
Christoph Koch and Paolo Petta

Models of Coordination 78
Robert Tolksdorf

Methodologies and Tools

From Analysis to Deployment: A Multi-agent Platform Survey 93
Pierre-Michel Ricordel and Yves Demazeau

Exploiting UML in the Design of Multi-agent Systems 106
Federico Bergenti and Agostino Poggi

Formal Specification and Prototyping of Multi-agent Systems 114
Vincent Hilaire, Abder Koukam, Pablo Gruer and Jean-Pierre Müller

Combining Software Components and Mobile Agents 128
Mercedes Amor, Mónica Pinto, Lidia Fuentes and José María Troya

Author Index 143

Engineering Social Order *

Cristiano Castelfranchi

National Research Council - Institute of Psychology
Division of "Artificial Intelligence, Cognitive and Interaction Modelling"
castel@ip.rm.cnr.it

Abstract. Social Order becomes a major problem in MAS and in computer mediated human interaction. After explaining the notions of Social Order and Social Control, I claim that there are multiple and complementary approaches to Social Order and to its engineering: all of them must be exploited. In computer science one try to solve this problem by rigid formalisation and rules, constraining infrastructures, security devices, etc. I think that a more socially oriented approach is also needed. My point is that Social Control – and in particular decentralised and autonomous Social Control – will be one of the most effective approaches.

1 The Framework: Social Order *vs* Social Control

This is an introductory paper. I mean that I will not propose any solution to the problem of social order in engineering cybersocieties: neither theoretical solutions and even less practical solutions. I want just to contribute to circumscribe and clarify the problem, identify relevant issues, and discuss some notions for a possible ontology in this domain.

I take a cognitive and social perspective, however I claim that this is relevant not only for the newborn *computational social sciences*, but for networked society and MAS. There is a dialectic relationship: on the one hand, in MAS and cybersocieties we should be inspired by human social phenomena, on the other hand, by computationally modelling social phenomena we should provide a better understanding of them.

In particular I try to understand what Social Order [1] is, and to describe different approaches to and strategies for Social Order, with special attention to Social Control

* This work has been and is being developed within the ***ALFEBIITE*** European Project**:** *A Logical Framework For Ethical Behaviour Between Infohabitants In The Information Trading Economy Of The Universal Information Ecosystem.* - IST- 1999-10298.

[1] The spreading identification between "social order" and cooperation is troublesome. I use here social order as “desirable”, good social order (from the point of view of an observer or designer, or from the point of view of the participants). However, more generally *social order* should be conceived as any form of systemic phenomenon or structure which is sufficiently stable, or better either self-organising and self-reproducing through the actions of the agents, or consciously orchestrated by (some of) them. Social order is neither necessarily

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 1-18, 2000.

and its means. Since the agents (either human or artificial) are relatively autonomous, act in an open world, on the basis of their subjective and limited points of view and for their own interests or goals, Social Order is a problem. There is no possibility of application for a pre-determined, "hardwired" or designed social order. Social order has to be continuously restored and adjusted, dynamically produced by and through the action of the agents themselves; this is why Social Control is necessary.

There are multiple and complementary approaches to Social Order and to its engineering: all of them must be exploited. In computer science one try to solve this problem by rigid formalisation and rules, constraining infrastructures, security devices, etc. I think that a more socially oriented approach is also needed. My point is that Social Control - and in particular decentralised and autonomous Social Control - will be one of the most effective approaches.

2 The Big Problem: Apocalypse Now

I feel that the main trouble of infosocieties, distributed computing, Agent-based paradigm, etc. will be -quite soon- that of the "social order" in the virtual or in artificial society, in the net, in MASs. Currently the problem is mainly perceived in terms of "security", and in terms of crisis, breakdowns, and overload, but it is more general. The problem is

> **how to obtain from local design and programming, and from local actions, interests, and views, *some desirable and relatively predictable/stable emergent result.***

This problem is particularly serious in *open* environments and MASs, or with heterogeneous and self-interested agents, where a simple organisational solution doesn't work.

This problem has several facets: Emergent computation and indirect programming [5,18]; reconciling individual and global goals [25,31]; the trade-off between initiative and control; etc.). Let me just sketch some of these perspectives on *the* problem.

2.1 Towards Social Computing: Programming (with) 'the Invisible Hand'?

Let me consider the problem from a computational and engineering perspective. It has been remarked how we are going towards a new "social" computational paradigm [19,20]. I believe that this should be taken in a radical way, where "social" does not means only organisation, roles, communication and interaction protocols, norms (and other forms of coordination and control); but it should be taken also in terms of spontaneous orders and self-organising structures. That is, one should consider the *emergent* character of computation in Agent-Based Computing.

In a sense, the current paradigm of computing is going beyond strict 'programming', and this is particularly true in the agent paradigm and in large and open MASs.

---

cooperative nor a "good" social function. Also systematic *dys-functions* (in Merton's terminology) are forms of social order [10]. See Section 3.

On the one hand the agents acquire more and more features such as:

- ***adaptivity***: either in the sense that they learn from their own experience and from previous stimuli; or in the sense that there may be some genetic recombination, mutation, and selection; or in the sense that they are reactive and opportunistic, able to adapt their goals and actions to local, unpredictable and evolving environments.
- ***autonomy and initiative*:** the agent takes care of the task/objective, by executing it when it finds an opportunity, and proactively, without the direct command or the direct control of the user; it is possible to delegate not only a specified action or task but also an objective to bring about in any way; and the agent will find its way on the basis of its own learning and adaptation, its own local knowledge, its own competence and reasoning, problem solving and discretion.
- ***distribution and decentralisation*:** MAS can be open and decentralised. It is neither established nor predictable which agent will be involved, which task it will adopt, and how it will execute or solve it. And during the execution the agent may remain open and reactive to incoming inputs and to the dynamics of its internal state (for example, resource shortage, or change of preferences). Assigned tasks are (in part) specified by the delegated agents: *nobody knows the complete plan*. Nobody entirely knows who delegated what to whom.

In other words, nobody will be able to specify *where*, *when*, *why*, *who* is running a given piece of the resulting computation (and especialli *how*). The actual computation is just emergent. Nobody directly wrote the program that is being executed. We are closer to Adam Smith's notion of 'the invisible hand' than to a model of a plan or a program as a pre-specified sequence of steps to be passively executed.

This is on my view the problem of ***Emergent Computation*** (EC) as it applies to DAI/MAS.

Forrest [18] presents the problem of Emergent Computation as follows:

> The idea that interactions among simple deterministic elements can produce *interesting and complex global behaviours* is well-accepted in sciences. However, the field of computing is oriented towards building systems that accomplish *specific tasks*, and emergent properties of complex systems are inherently difficult to predict and control. ... It is not obvious how architectures that have many interactions with often unpredictable and self-organising effects can be used effectively. The premise of EC is that interesting and useful computational systems can be constructed by exploiting interactions among agents.
>
> The important point is that the explicit instructions are at different (and lower) level than the phenomena of interest. There is a tension between low-level explicit computations and *direct programming*, and the patterns of their interaction'.
>
> Thus, there is some sort of 'indirect programming': *implementing computations indirectly as emergent patterns*.

Strangely enough, Forrest - following the fashion of that moment of opposing an anti-symbolic paradigm to the gofAI- does not mention DAI, AI agents or MAS at all; she just refers to connectionist models, cellular automata, biological and ALife models, and to the social sciences.

However, also higher-level components -complex AI agents, cognitive agents - give rise precisely to the same phenomenon (like humans!). More than this, I claim

that the 'central themes of EC' as identified by Todd [38] are among the most typical DAI/MAS issues.

Central themes of EC include in fact [38]:

- self-organisation, with no central authority to control the overall flow of computation;
- collective phenomena emerging from the interactions of locally-communicating autonomous agents;
- global cooperation among agents, to solve a common goal or share a common resource, being balanced against competition between them to create a more efficient overall system;
- learning and adaptation (and autonomous problem solving and negotiation) replacing direct programming for building working systems;
- dynamic system behaviour taking precedence over traditional AI static data structures.

In sum, Agent based computing, complex AI agents, and MASs are simply meeting the problems of human society: functions and 'the invisible hand'; the problem of a spontaneous emergent order, of beneficial self-organisation, of the impossibility of planning; but also the problem of *harmful self-organising behaviours.*

Let's look at the same problem from other perspectives.

2.2 Modelling Emergent and Unaware Cooperation among Intentional Agents

Macy [33] is right when he claims that *social cooperation does not need agents' understanding, agreement, contracts, rational planning, collective decisions*. There are forms of cooperation that are deliberated and based on some agreement (like a company, a team, an organised strike), and other forms of cooperation that are emergent: non contractual and even unaware. Modelling those forms is very important but my claim [2,10] is that it is important to model them not just among sub-cognitive agents [2] (using learning or selection of simple rules) [34,37], but also among cognitive and planning agents [3] whose behaviour is regulated by anticipatory representations (the "future"). Also *these agents cannot understand, predict, and govern all the global and compound effects of their actions at the collective level.* Some of these effects are self-reinforcing and self-organising.

---

[2] By "sub-cognitive" agents I mean agents whose behaviour is not regulated by an internal explicit representation of its purpose and by explicit beliefs. Sub-cognitive agents are for example simple neural-net agents, or mere reactive agents.

[3] Cognitive agents are agents whose actions are internally regulated by goals (goal-directed) and whose goals, decisions, and plans are based on beliefs. Both goals and beliefs are cognitive representations that can be internally generated, manipulated, and subject to inferences and reasoning. Since a cognitive agent may have more than one goal active in the same situation, it must have some form of choice/decision, based on some "reason" i.e. on some belief and evaluation. Notice that we use "goal" as the general family term for all motivational representations: from desires to intentions, from objectives to motives, from needs to ambitions, etc.

I argue that it is not sufficient putting deliberation and intentional action (with intended effects) together with some reactive or rule-based or associative layer/ behaviour, letting some unintended social function emerge from this layer, and letting the feedback of the unintended reinforcing effects operate on this layer. The real issue is precisely that *the intentional actions of the agents give rise to functional, unaware collective phenomena* (e.g., the division of labour), not (only) their unintentional behaviours. How to build unaware functions and cooperation on top of intentional actions and intended effects? How is it possible that positive results -thanks to their advantages- reinforce and reproduce the actions of intentional agents, and self-organise and reproduce themselves, without becoming simple intentions? [17]. This is the real theoretical challenge for reconciling emergence with cognition, intentional behaviour with social functions, planning agents with unaware cooperation. At the SimSoc'97 workshop in Cortona [10] I claimed that only agent based social simulation joint with AI models of agents can eventually solve this problem by formally modelling and simulating *at the same time* the individual minds and behaviours, the emerging collective action, structure or effect, and their feedback to shape minds and reproduce themselves.

I suggested that we need more complex forms of reinforcement learning not just based on classifiers, rules, associations, etc. but *operating on the cognitive representations governing the action, i.e. on beliefs and goals*.

My claim is precisely that "the consequences of the action, which may or may not have been consciously anticipated, modify the probability that the action will be repeated next time the input conditions are met" [6]:

> ***Functions*** *are just effects of the behaviour of the agents, that go beyond the intended effects (are not intended) and succeed in reproducing themselves because they reinforce the beliefs and the goals of the agents that caused that behaviour.*

2.3 Reconciling Individual with Global Goals

> "Typically, (intelligent) agents are assumed to pursue their own, individual goals. On this bases, a diversity of agent architectures (such as BDI), behavioural strategies (benevolent, antagonistic, etc.), and group formation models (joint intentions, coalition formation, and others) have been developed. All these approaches involve a bottom-up perspective. The existence of a collection of agents thus depends on a dynamic network of (in most cases: bilateral) individual commitments. Global behaviour (macro level) emerges from individual activities/interactions (micro level). .... In business environments, however, *the behaviour of the global system* (i.e., on the macro level) is important as well. Typical requirements concern system stability over time, a minimum level of predictability, an adequate relationship to (human-based) social systems, and a clear commitment to aims, strategies, tasks, and processes of enterprises. Introducing agents into business information systems thus requires to resolve this conflict of bottom up and top down oriented perspectives." [4] [31]

[4] I would say between an *individualist* and a *collectivist* perspective. Kirn proposes to deal with our problem with an organisational approach. This is quite traditional in MAS and is surely useful. However I claim in this paper that is largely insufficient.

This is another point of view on the same problem. It can be formulated as follows: How to reconcile individual rationality with group achievements?

Given goal-autonomous agents [5], basically there are two solutions to this problem of making the agent "sensible" to the collective interest:

* To use external incentives: such as prizes, punishments, redistribution of incomes, in general rewards, for ex. money (for example to make industries sensible to the environmental problem you can put taxes on pollution), so that the agent will find convenient - relatively to his/her selfish motives and utility - to do something for the group (to favour the group or to do as requested by the group). A special form of this is when incentives are not decided and planned by the group or authority; they are just the perceived advantages or disadvantages of conforming to or deviating from diffuse habits, conventions or rules. Conventions are in fact maintained by the fact that individual violation is dangerous, not convenient.
* To endow the agent with pro-social motives and attitudes (sympathy, group identity, altruism, etc.) either based on social emotions or not, either acquired (learning, socialisation) or inborn (by inheritance or design); in this case there is an intrinsic pro-group motivation. The agent is subjectively rational - although not economically rational- but ready to sacrifice [6].

Human societies use both these approaches [7]; this is not casual. We should experiment advantages and disadvantages of the two, to see on which domain and why one is better than the other.

Experimental *social simulation* can give a precious contribution to cope with this problem (for ex. [13,21,30]); but also good formal theories of spontaneous and non-spontaneous social order and of its mechanisms, and in particular theories of spontaneous and deliberated forms of "social control", will play a major role.

Clearly deontic stuff (norms, conventions, institutions, roles, commitments, obligations, rights, etc.) will have a large part in any "implementation" of social control mechanisms in virtual and artificial societies. However, on my view, this role will be improved by a more open and flexible view of deontic phenomena and by the embedding of them within the framework of social control and social order.

On the one hand, one should realised how social control and order has not only normative solutions (organisation and norm are not enough); on the other side -more

---

[5] i.e. self-motivated agents (self-interested but not necessarily selfish) that adopt goals only instrumentally to some goals of them (be these either selfish or altruistic) [1].

[6] This might be also objectively rational, adaptive: it depends on ecological and evolutionary factors.

[7] There is a sort of intermediate or double-face phenomenon which, depending on its use or modelling, can be considered as part either of (a) or of (b): *internal gratification or punishment* (like guilt). If the agent does something pro-social *in order to* avoid guilt, this is selfish motivation and behaviour (case a); but, notice that guilt feelings presuppose some pro-social goals like equity or norm respect! If, on the contrary, the agents act fairly and honestly just *for* these pro-social motives (not in order to avoid regret), and them feel guilty when violate (and guilt is just a learning device for socialisation) we are in case (b). The psychological notion of intrinsic motivation or internal reward mixes up the two. Both perspectives are realistic and even compatible.

important- one should account for a flexible view of normative behaviour, and for a gradual approach to norms from more informal and spontaneous phenomena (like spontaneous conventions and social norms, or spontaneous decentralised social control) to more institutionalised and formal forms of deontic regulation [9].

3 Approaches and Delusions

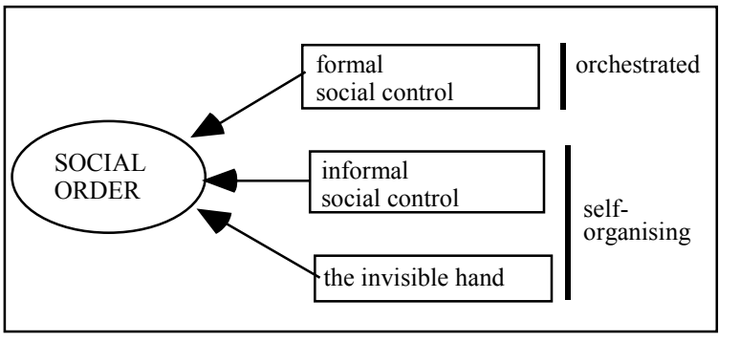
There are different "philosophies" about this very complex problem; different approaches and policies. For example:

- A **Coordination media and infrastructures** approach, where thanks to some infrastructures and media a desirable coordination among independent agents actions is obtained [16].
- A Social Control (SC) approach that is focused on sanctions, incentives, control, reputation, etc.
- An Organisational approach, relying on roles, division of labor, pre-established multi-agent plans, negotiation, agreements, etc.
- A Shared mind view of groups, teams, organisations where coordination is due to shared mental maps of the domain, task, and organisation, or to common knowledge of the individual minds [22].
- A Spontaneous Social Order approach where nobody can bear in mind, understand, monitor or plan the global resulting effects of the "invisible hand" (von Hayek) [24].

First, those approaches or views are not really conceptually independent of each other: frequently one partially overlaps with another, simply hiding some aspects or notion. For example coordination media are frequently rules and norms; and the same is true for organisational notions like "role" that are normative notions. All of them exploit very much communication. Second, in human groups -as I said- all these approaches are used, and Social Order is the result of both spontaneous dynamics and orchestrated and designed actions and constrains. It can be the result of SC and of other mechanisms like the "invisible hand", social influence and learning, socialisation, etc. But SC itself is ambiguous: it can be deliberated, official and institutional, or spontaneous, informal, and even unaware. I will completely put aside here education and social learning, and prosocial built-in motives (except for normative ones), although they play a very important role for shaping social order.

To be schematic, let's put at one extreme the merely self-organising forms unrelated to SC (ex.. market equilibrium) [8]; on the other extreme the deliberated and planned SC; in between there are forms of spontaneous, self-organising SC:

[8] Let's ignore here other factors of Social Order like constraining infrastructures for coordination (see section 4.).



In IT all of these approaches will prove to be useful. For the moment the most appealing solutions are:

- on the one hand, what I would like to call the "organisational" solution (pre-established roles, some hierarchies, clear responsibilities, etc.. This is more "designed", engineered, and rather reassuring!
- on the other hand, the "normative" or "deontic" solution, based on the formalisation of permissions, obligations, authorisation, delegation, etc. logically controllable in their coherence; 9
- finally, the strictly "economic" solution based on free rational agent dealing with utilities and incentives in some form of market.

The problem is much more complex, and – in my view – several other convergent solutions should be explored. However, I will consider here only one facet of the problem. My view is that

> *"normative" SC but spontaneous and decentralised has to play a major role both in cybersocieties and in artificial social systems.*

In IT there are some illusions about possible solutions.

The Illusion of Control: Security *Vs* Morality
The first, spontaneous approach of engineers and computer scientists to those issues is that of increasing security by certification, protocols, authentication, cryptography, central control, rigid rules, etc. Although some of these measures are surely useful and needed, as I said, I believe that the idea of a total control and a technical prevention against chaos, conflicts and deception in computers is unrealistic and even self-defeating in some case, like for building trust. Close to this illusion is the **formal-norm illusion** that I have already criticised (see later 5.).

The socio-anthropological illusion: let's embed technology in a social, moral and legal human context
In the area of information systems a perspective is already developing aimed at embedding the information system in a complex socio-cultural environment where there

[9] These two solutions can be strongly related one with the other, since one can give a normative interpretation and realisation of roles, hierarchies, and organisations.

are – on the top of the technical and security layer – other layers relative to legal aspects, social interaction, trust, and morality [23,32]. For sure this is a correct view. The new technology can properly work for human purposes only if integrated and situated in human morality, culture and law. However this is not enough.

For Intelligent Normative Supports (Agents)
I believe [9] that what is needed is some attempt

to "incorporate" part of these layers and issues in the technology itself.

Especially within the intelligent and autonomous agents paradigm, I believe that it is both possible and necessary to model these typically human and social notions. In order to effectively support human cooperation – which is strongly based on social, moral, and legal notions – computers must be able to model and "understand" at least partially what happens among the users. They should be able to manage – and then partially "understand"– for ex. permissions, obligations, power, roles, commitments, trust.

Moreover, to cope with the open, unpredictable social interaction and collective activity that will emerge among them, the artificial agents themselves should base this interaction on something like organisation, role, norms, etc. This is in fact what is happening in the domain of agents and MAS, where these topics are in the focus of theoretical and formal modelling, and of some implementation. (Just to give some example see DEON Ws, ModelAge project, the IP-CNR work; Jennings; Moses; Tennenholtz; Singh; Boman; etc.) [5,6,8,29,39].

4 Social Order

There are at least two notions of "Social Order".

Given a system of multiple entities, order is a *structure* of relationships

- among those entities or sub-units,
- of each of them with the whole, and
- with the environment of the system.

Such a trim, structure or pattern is "regular". "Regular" means either:
α) corresponding to a given "regularity", i.e., to a norm in statistical or previsional terms; or
β) corresponding to some rule (Latin: *regula*), i.e., some norm or standard in a normative/deontic sense.

The α–meaning is broader and weaker: order is just a recurrent or stable emergent pattern. The β–meaning is stronger: the order is "intended" or at least desirable (from the point of view of some agent) or "functional". In any case, it is not simply accidental but finalistic/teleonomic. I will use here β–meaning.

Social Order is a non-accidental and non chaotic (thus, relatively predictable, repeated and stable) pattern of interactions in a given system of interfering agents,

such that it allows the satisfaction of the interests of some agent A (or avoids their damage) [10]

Thus Social Order is relative to some system of interests or points of view, that makes it good or bad, desirable or undesirable. This point of view or reference can be:

- a single lay agent or group (internal or external to the order: observer);
- an agent with a special role: an authority, able (at least partially) to orient the system towards such an order;
- a shared goal, value, that is good for everybody or for most of the members (for example to reduce or avoid accidents; to guarantee the group survival; etc.);
- a real "common goal", in a strictly cooperative group.

Dynamic Social Order occurs when the stable macro-pattern or equilibrium is maintained thanks to *an incessant local (micro) activity* of its units, able to restore or reproduce the desired features. The global stability is due to local instability. Social order is very dynamic, especially with autonomous agents.

Given this notion of Social Order, let us examine some approaches and means for it.

5 Coordination Media and Infrastructures

Both "coordination" and "coordination media" are rather vague notions. "Coordination" can cover any good emerging social structure, any "order". I would like to use it not as a synonym of Social Order but in a more strict and delimited way [7].

"Coordination media" is also too broad. It can cover any device used/useful for coordination: communication of any kind, norms, (shared) mental models, representations and maps, roles, plans, trust, etc.

I do not find so useful such a broad set/container. I prefer to focus on more specific notions, for example norms (see later) or *constraining infrastructures* in a strict sense.

Virtuous coordination among the individual autonomous activities (producing a desirable global effect) is obtained sometimes thanks to coordination infrastructures, like for example the physical *barriers* for traffic in the cities (guard-rails, walls, sidewalks, etc.) or like corrals for animals, etc. or like accessible or non-accessible blackboard structures, or pre-established communication channels among computational agents.

What characterise this means for coordination or better for SO, is the fact that

there is a *practical impossibility* for the agents to deviate from the desired behaviour: the action is *externally constrained* by its execution or success *preconditions.*

I propose to call them: *constraining infrastructures* or *barriers*.

[10] The problem of social functions and functional order is far more complicated [10]. Let's put it aside here.

Every action (either complex like a plan, or elementary) has certain preconditions for being materially executed or for succeeding. For example, the action of "switching the light on" has as its execution preconditions that there is an accessible and reachable switch, that it works, that I can move my arm appropriately, etc. Condition for success is that there is power in the cables, that there is a working lamp; those cables are connected, etc. If some of these conditions do not hold, even though I correctly perform the motor part of my action (switching the switch), I will not switch the light on.

By establishing certain "material" conditions in the environment that the agent is not able to change, I prevent it from doing a given action (or even from attempting to do it), and at the same time -since it is motivated to achieve a given goal- I channel its behaviour in a given direction (towards the remaining practicable solutions) i.e.; towards what is practically permitted. Sometimes the behaviour is oriented towards the only remaining possible move, which becomes "obligatory" since there are no degrees of freedom for the agent. For example, if you put me in a tube, and I want to go out, and you block one access (my way in), I'm *obliged* or better constrained to go straight head, as you want.

Barriers can be physical for physical agents (in the sense that the agent meets them physically in attempting to execute the action and failing). But they can also be non-physical. If the agent has no mental alternatives to that behaviour, if it does not "choose" and "prefer", its behaviour is constrained by a cognitive *practical impossibility* to do differently.

This is why true normative constraints (rules and conventions) are different: agents are free to obey to or violate them. [11]

The very difference can be characterised in the following way. Considering the **architecture** of an autonomous cognitive agent [4,12] and how the goal-directed and belief-based intentional action is generated, we can say that:

> the conformity of the behavior to the expected/desired standard, or better the *elimination of the undesirable behaviors* can be obtained (at least) in three very different ways:
>
> i) by means of the beliefs relative to **practical impossibility**: "I cannot do" "I'm not able" ("Necessary conditions are not there and I cannot create them"). "*Barriers*" exploit this path.
>
> ii) by means of **learning** from frustrations or rewards.
> iii) by means of the beliefs relative to **utility and preference**: "it is not convenient for me" "too high risks"; "too high costs".

Since those beliefs determine the persistence or the abandon of a given goal (at the desire or at the intention level of processing) giving/changing those beliefs we create the practical or the decision impossibility to perform an undesirable behaviour [4].

Another important distinction, more general and applicable also to sub-cognitive rule-based agents, is between

---

[11] A lot of what is called "infrastructure" are in fact norms, signals/messages to the agents (for ex. semaphores). Rules/norms entail in fact messages for giving the commands to the agents.

- impossibility due to the internal stuff, to an internal lack of power of the agent (for example, the absence of a rule, of a skill, or the rule repertoire); and
- impossibility ascribable to external conditions and circumstances.

In such a framework one could generalise a notion of **"infrastructure"** as a structure - either physical (for physical actions) or informational (for communication and virtual actions)- which is *external* to the agents and to the actions but constrains the behaviour and its individual or collective results. In other words, "infrastructure" should be and environmental notion.

5.1 The Role of the Environment

Let me stress how important is to understand the role of the environment in coordination, and in general in (collective) action [36].

Agents act in a shared environment and the environment plays a crucial role in their interaction and coordination. More precisely, *Agents interact and communicate **through** the environment* [7,36]. Even more basically, any action is inter-action with some environment [7]: the result of the action is interactive and depends on the environment. Any action must -at least in part- be 'delegated' to the environment and executed by the external causal processes. The agent relies upon the environment. The result of the action depends not only on the agent's intention and control, but also on unpredictable and uncertain environmental dynamics. This holds not only for the physical environment but even more for the social environment and the result of actions in MA contexts that are defined by their high 'interference' [7].

Communication exploit the environment not only in the case of the so called "stygmergic communication", but also in normal messages exchange, since messages are propagated through the environment and the external infrastructures. The same holds for Coordination; either it is based on communication or it does not use communication but uses perception of the other's behaviours and environmental effects.

Thus, exploiting, organising, and design the environment is a fundamental approach for engineering SO.

6 The Feedback Problem in Social Order

A very relevant problem (and an essential means) is how to **model the *feedback*** that must arrive at the local agent level (and enter its cognitive processes and decision, or its learning, or its reactive response) in order for it to adjust its behaviour in the "right" way (relative to the desired global result). I mean the feedback from the global emerging structure to the individual agents.

Several theoretical possibilities should be explored.

a) the agents have/receive a representation of the global resulting structure or phenomenon

a1) they care for the global result and regulate their behaviours to achieve a specific global structure. This is the case for ex. of 'ring-a-ring-o'-roses' [12] where the agents adjust their own behaviour by checking the global circular structure; they can also communicate with each other and prescribe the others certain behaviours in order to co-ordinate with each other.

a2) they receive the global result information but they adjust to it for their own purposes. For example, there is a traffic jam (or they receive the warming that there is a clogging on the highway) and they change their own route to avoid the row They do not care for reducing or not producing the jam, they just care of their own goal, but they adjust on a global-structure feedback.

b) The agents do not have a representation of the global result (they would not be able to understand it, or it is too complex and costly, etc.); they just receive some partial information about or feedback from the global result and they adjust their personal behaviour (for their personal goals) to this feedback.

The most celebrated and beautiful example is from economics: the theory of price as information about a general market equilibrium: price is the necessary and sufficient information the agent has to locally know in order to decide; by this information and adjusted local decisions the global equilibrium (that nobody calculates or intends) is achieved.

However, these are just three basic possibilities; indeed a systematic theory of different kinds and functions of that feedback from the global to the local is needed.

Let me now focus on what I believe to be the most effective and important approach to SO, i.e. SC.

7 Social Control Approach to *Dynamic Social Order*

As I said at the beginning, precisely because agents are relatively autonomous, act in an open world, on the basis of their subjective and limited points of view and for their own interests or goals, Social Order is a problem. There is no possibility of application for a pre-determined, "hardwired" or designed social order. Social order has to be continuously restored and adjusted, dynamically produced by and through the action of the agents themselves; this is why social control is necessary.

7.1 Social Control (SC)

Let's take SC in a strict sense, not as any form of *social influence* on agents and of *socialisation* (although obviously both can strongly contribute to social order). Let's consider SC only as the process through which, if/when an individual or a group derogates from the expected and prescribed degree of obedience to a norm, its behaviour is led back to that degree of conformity [26]. Social control is a reaction to deviant behaviour, and is strongly related to the notion of 'sanction' [35].

[12] I take this nice example from a talk by Huhns at AOIS'99.

However, let's also consider pro-active actions, prevention from deviation and reinforcement of correct behaviour, and then also "positive" sanctions, social approval (also an implying implicit message/disapproval for deviant people) as part of SC.

I consider SC ***any action of an agent aimed at (intended or destined to) enforcing the conformity of the behaviour of another agent to some social norm*** (in a broad sense, including social roles, conventions, social commitments, etc.). [13]

Thus, not all socialisation is SC (there are also other purposes and functions in socialisation); not all social influence is SC. However, not only negative sanctions and post-hoc corrective interventions are SC.

As defined SC strictly presupposes:

- some informal or formal, implicit or explicit, social or legal norm or convention in the society/group,
- the possibility for the agent to deviate from it and to be led back (through psychological means) to the right behaviour. SC presupposes autonomous agents, and possibly decision-makers and normative agents [5,8,39], or at least some learning capability based on social rewards.

There are different forms of SC. It is useful to distinguish at least: deliberative/intended *Vs* unintended/functional SC; and centralised *Vs* decentralised SC.

	intended	unintended
centralized	A	B
decentralized	C	D

My claim (see also [15,8,30]) [14] is that A is **not** the prototypical or the most efficacious form of SC in open and dynamic MAS or societies; a fundamental role is played

[13] I am close to Johnson's [27] view that SC consists in the action of all those mechanisms that neutralise deviant tendencies, either by preventing deviant behaviours, or -more important- by monitoring and inverting the motivational factors that can produce the deviant behaviour. I only find too general the idea of SC as all the mechanisms that actually produce those effects. I prefer to restrict SC to those mechanisms *aimed* at producing those effects (functions) [3].

[14] Kaminka and Tambe [30] present interesting results. They claim that Agents in dynamic multi-agent environments must monitor their peers to execute individual and group plans. They show that a centralised scheme using a complex algorithm trades correctness for completeness and requires monitoring all teammates. By contrast, a simple distributed teamwork monitoring algorithm results in correct and complete detection of teamwork failures, despite relying on limited, uncertain knowledge, and monitoring only key agents in a team.

by C and D, i.e., the spontaneous, bottom-up normative intervention of the distributed individuals, either conscious of their effects and intending to make another conform to the norms, or not intentionally oriented to this but in fact *functional* to this. For example, when Ag1 – a victim – complains for its pain or damage or aggressively reacts against Ag2 – the culprit – one of the effects and of the functions of these reactions is precisely to make Ag2's behaviour conform to norms. Analogously, people just moving around in the city and observing the other people (what they are doing) are in fact exerting a form of non-voluntary SC (and this behaviour actually reduces crimes).

Decentralised and in particular unintended SC (box D) are specially important in order to understand that also SC has some unplanned form and contributes to a merely emergent and self-organised Social Order.

Moreover, not only the monitoring and the intervention can be bottom-up and spontaneous (while the norm can still be formal and official); all normative bonds can be bottom-up, informal and spontaneous: the creation of conventions and norms, the establishment of rights, duties, permissions, etc. [15] [3,9].

Given the features of infosocieties, computer mediated interactions, and of heterogeneous MASs my point is that we should understand and model these forms of SC, in order to engineer them in cybersociety. Formal and top-down forms of control cannot be enough.

7.2 Three Postulates

In sum, my analysis is framed by the following general assumptions:

- Social Order does not only depend on SC and is not reducible to SC

Other mechanisms (for ex. the so-called "invisible hand", the emergent result of self-interest) produce social order, like the natural division of labour, or the market equilibrium. We do not define in fact social order as norm conformity. This is too strong. Even "desirable" social order is not simply or necessary norm-conformity. However, in this paper we have been mainly interested in a SC approach to Social Order, not in other mechanisms for emergent, self-organising Social Order.

- SC is not due to norms only and is not reducible to them [16]

---

[15] Also the classical distinction between *external* and *internal* control (self-control) [26] is very important. True norms are aimed in fact at the internal control by the addressee itself as a cognitive deliberative agent, able to understand a norm as such and adopt it [12]. Even more, norms are aimed at being adopted for specific reasons by the agents. The use of external control and sanctions is only a sub-ideal situation and obligation [8]. In artificial agents internal control is possible with real decision makers, be either norm-sensible agents or utility-sensible agents. Although self-control is for sure very important for Social Order and also for conformity to norms, I do not want to consider here it as a form of SC (see in fact our definition in terms of two different agents).

[16] Although – as I said – SC always presupposes some form of norm or convention, the norm itself is not the only means and strategy for obtaining conformity to norm. I can obtain conformity also by agents that lack any normative mind and understanding.

There are various forms of and instruments for SC, beyond the explicit use of (social and legal) norms; ex. imitation; incentives; learning; etc.

- Normative means are not only based on or reducible to formal, top down, institutional norms

The *informal* normative relationships, *the Micro/ Bottom-Up/ Decentralised/ Spontaneous Normative Social Control* are very important also for artificial systems.

Recapitulation

- Social Order does coincide with either Social Control or Organisation; there are multiple and complementary approaches to Social Order and to its engineering: all of them must be exploited (thus modelled) in artificial systems.
- It is not possible to really "design" societies and the Social Order; more precisely it is possible only in some cases like in certain kinds of organisations; it is necessary to design only indirectly, i.e. to design frameworks, constraints and conditions (both internal ad external to the agents) in which the society can spontaneously self-organise and realise the desirable global effects.
- There is some illusion in computer science about solving this problem by rigid formalisation and rules, constraining infrastructures, security devices, etc. and there is scepticism or irritation towards more soft and "social" approaches, that leave more room to spontaneous emergence, or to decentralised control, or to normative "stuff" which is not externally imposed but internally managed by the agents.
- Social modelling will be the principal solution; it should leave some flexibility and try to deal with emergent and spontaneous forms of organisation; but there are serious problems like that of modelling the feedback from the global results to the local/individual layer.
- Social Control (and in particular decentralised and autonomous SC) will be one of the most effective approaches.

References

1. Binmore, K., Castelfranchi, C., Doran, J. and Wooldridge, M. Rationality in Multi-Agent Systems. The Knowledge Engineering Review, Vol. 13:3, 1998, 309-14.
2. Castelfranchi, C., and R. Conte. Emergent functionalitiy among intelligent systems: Cooperation within and without minds. *AI & Society, 6,* 78-93, 1992.
3. Castelfranchi, C., Commitment: from intentions to groups and organizations. In *Proceedings of ICMAS'96*, S. Francisco, June 1996, AAAI-MIT Press.
4. Castelfranchi, C. Reasons: Belief Support and Goal Dynamics. *Mathware & Soft Computing, 3*. pp. 233-47.

5. Castelfranchi,C. Emergence and Cognition: Towards a Synthetic Paradigm in AI and Cognitive Science. In H. Coelho (Ed.) *Progress in Artificial Intelligence* - IBERAMIA 98, Springer, LNAI1484, Berlin, 1998 13-26.
6. Castelfranchi, C. Through the minds of the agents, Journal of Artificial Societies and Social Simulation, 1(1), 1998.
7. Castelfranchi, C., Modelling Social Action for AI Agents. *Artificial Intelligence*, 1999.
8. Castelfranchi, C., Dignum, F., Jonker, C., Treur, J. (1999) Deliberate Normative Agents: Principles and Architecture. *ATAL'99,* Boston.
9. Castelfranchi, C. Formalising the Informal? (invited talk). *DEON'00*, Touluse.
10. Castelfranchi, C. The theory of social functions. Challenges for multi-agent-based social simulation and multi-agent learning. (invited talk at *SimSoc'97*, Cortona, Italy) to appear in *Cognitive Systems*, Springer.
11. Castelfranchi, C. and Tan, Y.H. (eds.) (in press) "Trust, Deception and Fraud in Artificial Societies" Kluwer.
12. Conte,R. and Castelfranchi, C. *Cognitive and Social Action*, UCL Press, London, 1995.
13. Conte, R., Hegselmann, R. and Terna, P. (eds) Studies in social simulation. Berlin, Springer, 1997.
14. Conte, R. e Castelfranchi, C. (1998) From conventions to prescriptions. Towards a unified theory of norms. *AI&Law*, 1998, 3.
15. Conte, R., Castelfranchi, C., Dignum, F. Autonomous Norm Acceptance. In J. Mueller (ed) *Proceedings of the 5th International workshop on Agent Theories Architectures and Languages,* Paris, 4-7 July, 1998 (in press).
16. Denti, E., Omicini, A., Toschi, V. Coordination technology for the development of MAS on the Web. In E. Lamma and P. Mello (eds.) AI*IA '99 Congress, Bologna, Pitagora Editrice. pag. 29-38.
17. Elster, J., Marxism, functionalism and game-theory: the case for methodological individualism. Theory and Society 11, 453-81.
18. Forrest, S., (ed.).*Emergent computation*. Cambridge, Mass. MIT Press.1990.
19. Gasser, L.,. Social conceptions of knowledge and action: DAI foundations and open systems semantics. *Artificial Intelligence* 47: 107-138.
20. Gasser, L.,. Invited talk at *Autonomous Agents'98*, Minneapoli May 1998.
21. Gilbert, N., and Conte, R. (eds) Artificial Societies: the computer simulation of social life. London, UCL Press, 1995.
22. Grosz, B., Collaborative Systems. *AI Magazine*, summer 1996, 67-85.
23. Hartmann, A. (1995) "Comprehensive information technology security: A new approach to respond ethical and social issues surrounding information security in the 21st century". In IFIP TCI 11 Intern. Conf. of Information Security.
24. Hayek, F. A., The result of human action but not of human design. In *Studies in Philosophy, Politics and Economics*, Routledge & Kegan, London, 1967.
25. Hogg L. M. and Jennings N. R. (1997) : Socially Rational Agents, in Proc. AAAI Fallsymposium on Socially Intelligent Agents, Boston, Mass., November 8-10, 61-63.
26. Homans, G.C., *The Human Group*, N.Y. 1950.
27. Johnson, H.M. *Sociology: a systematic introduction*. N.Y. 1960.

28. Jennings, N. R.,. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review* 3, 1993: 223-50.
29. Jones, A.J.I. & Sergot, M. 1996. A Formal Characterisation of Institutionalised Power. *Journal of the Interest Group in Pure and Applied Logics*, 4(3): 427-45.
30. Kaminka, G.A. and Tambe, M. (2000) "Robust Agent Teams via Socially-Attentive Monitoring", Journal of Artificial Intelligence, Volume 12, pages 105-147.
31. Kirn, S. Invited Talk. *AOIS'99*, Hidelberg.
32. Leiwo, J. and Heikkuri, S. (1996) An Analysis of Ethics Foundations of Information Security in Distributed Systems, TR. Nokia TeleC, Helsinki.
33. Macy, M. Social Order in Artificial Worlds, In *JASSS* I, 1 - *Journal of Artificial Societies and Social Simulation*, vol.1, no.1, 1998. http://www.soc.surrey.ac.uk/JASSS/1.html.
34. Mataric, M.,. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *Simulation of Adaptive Behavior 2*. MIT Press. Cambridge, 1992.
35. Parsons, T. *The Social System*, Glencoe, Ill. 1951.
36. Parunak, H. V. N., S. Brueckner, J. Sauter, R. Matthews. Distinghuishing Environmental and Agent Dynamics: A Case study in Abstraction and Alternative Modeling Technologies. In this volume, 2000.
37. Steels, L., Cooperation between distributed agents through self-organization. In Y. Demazeau & J.P. Mueller (eds.) *Decentralized AI* North-Holland, Elsevier, 1990.
38. Todd. P.M. Book Review of S. Forrest 'Emergent Computation'. Artificial Intelligence, 60, 1993, 171-183.
39. Verhagen, H. *Normative Autonomous Agents*. PhD. Thesis, University of Stockolm, May, 2000.
40. Wagner, G. (1997) "Multi-Level Security in Multiagent Systems", in P. Kandzia and M. Klusch (Eds), *Cooperative Information Agents*, Springer LNAI 1202, 1997, 272-285.
41. Wooldridge M.J. and Jennings N.R. (Eds.) 1995 *Intelligent Agents: Theories, Architectures, and Languages.* LNAI 890, Springer-Verlag, Heidelberg, Germany.

Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies

H. Van Dyke Parunak, Sven Brueckner, John Sauter, and Robert S. Matthews

ERIM Center for Electronic Commerce,
PO Box 134001, Ann Arbor, MI 48113-4001 USA
{vparunak,sbrueckner,jsauter,rmatthews}@erim.org

Abstract. Two factors can confound the interpretation of an MAS application or model. First, the MAS's dynamics interact in complex ways with those of its environment, and agent engineers need to distinguish the two. Second, "mean field" approximations of the behavior of the system may be useful for qualitative examination of the dynamics, but can differ surprisingly from the behavior that emerges from the interactions of discrete agents. This paper examines these effects in the context of a project applying synthetic pheromones for agent-based control to a military air operations scenario.

1 Introduction

Modern military operations can overwhelm a commander. The information available from satellite and other sensors floods conventional analysis methods. Enemy forces using advanced technology can hide or change location faster than conventional planning cycles can respond, and coordinating central orders across thousands of friendly resources can slow response even further. These characteristics are hallmarks of situations for which agent-based systems are particularly suitable.

ADAPTIV (Adaptive control of Distributed Agents through Pheromone Techniques and Interactive Visualization) applies fine-grained agent techniques to the control of air resources charged with defending a friendly region from enemy attack. Intelligence on the location and strength of enemy ("Red") resources leads to the deposit of synthetic pheromones (SP's) [3, 14, 16] in a spatial model of the battlespace. The propagation and evaporation of SP's model the uncertainty in the available intelligence, and generate a flow field that guides friendly ("Blue") units.

In our experiments with these mechanisms, Red moves ground troops under cover of air defense toward Blue's territory. Blue's bombers can attack the ground troops, but are vulnerable to Red air defenses. Blue's fighters can suppress air defenses and thus protect bombers. Our initial experiments with SP's in this scenario yield surprisingly complex variations in outcome as we vary the distribution of resources on each side. Some of this variation results from the strategies being executed by our agents, but much is due to the complexity of the game rules. That is, the environment

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 19-33, 2000.

is not passive, but active (a point made elsewhere in this volume [4, 12]), and its actions can contribute to the outcome of the model. To separate the two effects, we have applied three successive abstractions to the initial experimental setting, first neutralizing the effect of Blue strategies, next removing the spatial structure of the game entirely, and finally abstracting away from the individual units with a mean field approximation to the combat rules. We develop two morals that go beyond SP's. First, experimental abstraction is a useful (even necessary) technique for understanding the relative effects of agents and environment (or to use the vocabulary of control theory, of the controller and the plant). Second, modeling technology can distort the picture in ways to which the analyst must be sensitive.

Section 2 briefly motivates and reviews SP technology, explaining our mechanisms and comparing our approach with other related work. Section 3 describes the problem domain, our experimental scenario, and initial results. Section 4 applies three successive abstractions to this scenario in an effort to distinguish environmental and agent dynamics. Section 5 discusses our experience and summarizes key insights.

2 Synthetic Pheromones for Multi-agent Coordination

Many applied problems require that entities move from one location to another under certain constraints. These problems have traditionally been addressed with centralized planning and control mechanisms. In highly dynamic domains such as combat management, central coordination may not permit timely response, and the central controller is a vulnerability that can place the entire operation at risk.

Negotiation schemes from recent MAS research enable the entities to maintain such constraints. These mechanisms effectively decentralize system control, but can require sophisticated inter-agent communication and significant processing within individual agents. There are several motives for seeking simpler mechanisms.

- The resources needed for conventional negotiation restrict the deployment of these techniques to highly numerous, relatively inexpensive entities such as boxes of field rations or seat assemblies en route from supplier to final assembly.
- The dynamics of classical DAI mechanisms are poorly understood, and in some cases may become intractable, leading to inadequate performance.
- The complexity of designing interlocking protocols so that they all work correctly is often non-trivial.
- In a military context, high semantic content in inter-agent messages is a point of vulnerability to evesdropping.

Insect colonies perform sophisticated motion coordination and control using neither central coordination nor direct agent-to-agent communication. We are developing mechanisms that mimic their behavior in engineered systems.

2.1 Insect Examples

Insects perform impressive feats of coordination without direct inter-agent coordination, by depositing pheromones (chemical scent markers) in the environment

and then sensing them [14]. For example, ants construct networks of paths that connect their nests with available food sources. Mathematically, these networks form minimum spanning trees [9], minimizing the energy ants expend in bringing food into the nest. Graph theory offers algorithms for computing minimum spanning trees, but ants do not use conventional algorithms. Instead, this globally optimal structure emerges as individual ants wander, preferentially following food pheromones and dropping nest pheromones if they are not holding food, and following next pheromones while dropping food pheromones if they are holding food.

Brownian motion brings the ant arbitrarily close to every point in the plane. As long as the separation between nest and food is small compared with the ant's range, a wandering ant will find food if there is any, and a food-carrying ant will find the nest.

Because only food-carrying ants drop food pheromone, and because ants carry food only after picking it up at a source, all food pheromone paths lead to food. Because only empty ants drop nest pheromone, and because all empty ants originate at the nest, all nest pheromone paths lead home. Because pheromones evaporate, paths to depleted food sources disappear, as do paths laid down by ants that get lost.

The initial path will not be straight, but the tendency of ants to wander even in the presence of pheromones will generate short-cuts across meanders. Overlapping pheromone paths tend to merge together into a trace that becomes straighter the more it is used. The character of the resulting network as a minimal spanning tree is not intuitively obvious from the individual behaviors, but emerges from the emulation.

2.2 Mechanisms

These behaviors manifest two mechanisms that we seek to emulate: stochastic movement, and pheromones.

Stochastic search is the ultimate "weak method." By itself, it fails under combinatorial explosion. However, when tempered with other mechanisms (such as temperature in simulated annealing, or crossover in genetic algorithms, or local alignment with other agents in particle swarm optimization), it is ubiquitous in practical weak methods. It is an essential component in most models of insect behavior, and we will incorporate it in our model.

The real world provides three operations on chemical pheromones that support purposive insect actions. It *aggregates* deposits from individual agents (providing integration of information across multiple agents and through time), *evaporates* them over time (thus forgetting obsolete information and avoiding overloading), and *diffuses* them to nearby places (generating a gradient that agents can follow). A *pheromone infrastructure* [3] is a software environment that supports these operations. It consists of a network of places over which agents move, and between which pheromones propagate. At any moment in time, an agent is located at a specific. This place offers it a number of services, including the ability deposit pheromones and query the strength of pheromones deposited by others.

These techniques can be applied to real-world problems in two ways. Sometimes actual physical entities can move immediately in response to pheromones. If physical entities cannot tolerate the intrinsic stochasticity, the emergent behavior of a population of virtual agents can be consulted to guide the real entities. These simple mechanisms can run extremely rapidly, permitting simulation in faster than real-time.

2.3 Comparison with Other Research

The potential of insect models and similar mechanisms for multi-agent coordination and control is receiving increasing attention [1, 14]. Ferber [8] and Drogoul [6] offer theoretical discussions with simple applications, and Drogoul [7] shows how these techniques can play a credible game of chess. White [22] constructs a synthetic chemistry through which agents interact.

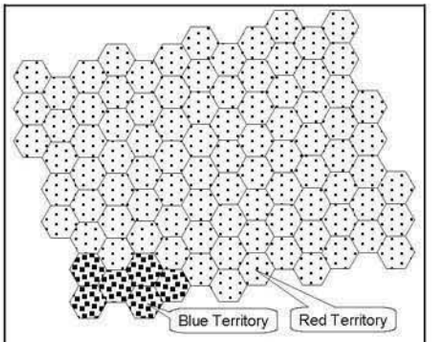


Fig. 1. The SEADy Storm Board

Pheromone-based techniques have been developed for routing telecommunications packets [2] and moving physical entities [17]. The latter work appealed to a neural backpropagation model for its antecedents. However, the accumulation of weights on frequently activated links through backpropagation has many formal similarities to the accumulation of pheromones on well-traveled paths. Steels proposed similar mechanisms for coordinating small robots used in exploring remote planets [21]. These techniques are applicable to a range of optimization problems including the traveling salesperson problem and the quadratic assignment problem [5].

Our approach is distinct in three ways.

1. We extend SP's with mechanisms that permit human overseers to monitor and influence them as they operate.
2. We hybridize SP's subsymbolic reasoning with symbolic processing.
3. We give special attention to tools and methods for engineering SP's for real-world problems. This paper's theme reflects this distinctive.

3 The SEADy Storm Experimental Context

First we summarize the experimental scenario. Then we describe the behavior of our agents, and show the results from our initial experiments.

3.1 The SEADy Storm Game

The SEADy Storm war game [11] exercises technologies for controlling air tasking orders. The battlespace is a hexagonal grid of 50-km sectors (Figure 1). Friendly (Blue) forces defend against invading Red forces that include ground troops (GT's) that are trying to invade the Blue territory, and air defense units (AD's) that

	Move	Attack	Wait
AD	Relocate	Fire (on any Blue aircraft)	Hide Deceive
GT	Advance		Hide
SEAD	NewSectors	AttackAD	Rest
BMB	NewSectors	AttackAD AttackGT	Rest

Table 1. Unit Commands in SEADy Storm

protect the GT's from Blue attack. Blue has bombers (BMB's) to stop the GT's,, and fighters to suppress enemy air defenses (SEAD's).

Each class of unit has a set of commands from which it periodically chooses. Ground-based units (GT and AD) choose a new command once every 12 hours, while air units (BMB and SEAD) choose once every five minutes, reflecting unit velocities. The commands fall into three categories (Table 1). GT cannot attack Blue forces, but can damage BMB's if they attack GT.

Blue can attack AD and GT when they are moving or attacking, and AD may attack any Blue forces that are not moving or waiting. Each unit has a strength that is reduced by combat. The strength of the battling units, together with nine outcome rules, determine the outcome of such engagements. Informally, the first five rules are:

1. Fatigue: The farther Blue flies, the weaker it gets.
2. Deception: Blue strength decreases for each AD in the same sector that is hiding.
3. Maintenance: Blue strength decreases if units do not rest on a regular basis.
4. Surprise: The effectiveness of an AD attack doubles the first shift after the unit does something other than attack.
5. Cover: BMB losses are greater if the BMB is not accompanied by enough SEAD.

Rules 6-9 specify the percentage losses in strength for the units engaged in a battle, on the basis of the command they are currently executing. For example, Rule 9, in full detail, states: "If BMB does "AttackGT" and GT does "Advance": a GT unit loses 10% for each BMB unit per shift; a BMB unit loses 2% per GT unit per shift."

3.2 The ADAPTIV Mechanisms

ADAPTIV controls Blue operations with pheromone techniques. Intelligence reports on Red locations deposit SP's in a spatial model reflecting the hexagonal grid. The experiments reported here manipulate a package of BMB and SEAD as a unit. When a unit is eligible for a new command, it selects with equal probability from its possible commands. If it selects a movement command, its movement depends on its class and the SP's it senses in its own and the adjacent sectors. A unit "follows" the pheromone field using a roulette wheel weighted by the strength of the SP's:

- the SEAD-BMB package follows GT pheromones,
- AD units follow a product of BMB and GT pheromones, thus seeking out BMB's that are threatening GT's, and
- GT units move randomly, with higher weights given to south-westerly neighbors.

An experiment runs for 1200 simulated hours. Metrics include the total Red strength that has reached Blue territory ("Red in Blue" or "RinB") and the surviving percentages of each class of unit. We run each configuration of parameters eleven times with different random seeds, and report medians for each configuration.

3.3 EXP: Experimental Results

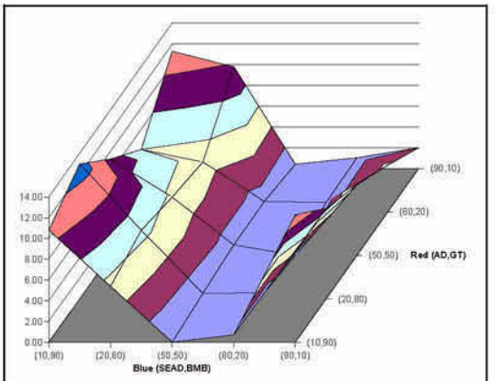


Fig. 2. Red Strength in Blue as a function of force composition (EXP)

The primary parameter explored in the experiments reported here is the proportion of SEAD in the Blue military, and of AD in the Red military. Each side began with a 100 units, each with unit strength, and 10%, 20%, 50%, 80%, or 90% of SEAD or AD. The uneven spacing reflects a basic statistical intuition that interesting behaviors tend to be concentrated toward the extremes of percentage-based parameters. In current military doctrine, 50% is an upper limit on both AD and SEAD. We explore higher values simply to characterize the behavioral space of our mechanisms.

The central outcome is total Red strength in Blue territory at the end of the run (Figure 2). The landscape shows several interesting features, including

- a "valley" of Blue dominance for all Red ratios when Blue SEAD is between 50% and 80%, with slightly increasing Red success as the AD proportion increases;
- clear Red dominance for lower SEAD/BMB ratios, decreasing as SEAD increases;
- a surprising increase in Red success for the high SEAD and low AD levels.

Figure 3 shows the surviving percentages of each class of unit at the end of the run. AD, GT, and BMB reflect the main features of the topology. The increase in Red effectiveness for high SEAD appears to be due to a drop in BMB survival in this region, a surprising effect since BMB's have strong SEAD protection here.

These interesting and non-trivial dynamics have two sources: the pheromone-based movement of the resources, and the outcome rules that define the scenario. For example, Red superiority at low SEAD ratios is directly related to Rule 5, which places a particularly heavy penalty on Blue packages that do not have at least one SEAD for every two BMB's. This rule induces a threshold nonlinearity at SEAD/BMB 33/67, which marks the edge of the Blue valley in other runs (not shown here) that explore the parameter space in more detail. However good Blue's pheromone algorithms are at finding and targeting Red troops, Rule 5 will impose a performance cliff along this parameter.

4 Successive Abstractions

To understand the contribution of our mechanisms, we must distinguish their dynamics from those of their environment. We abstract away successive details of our

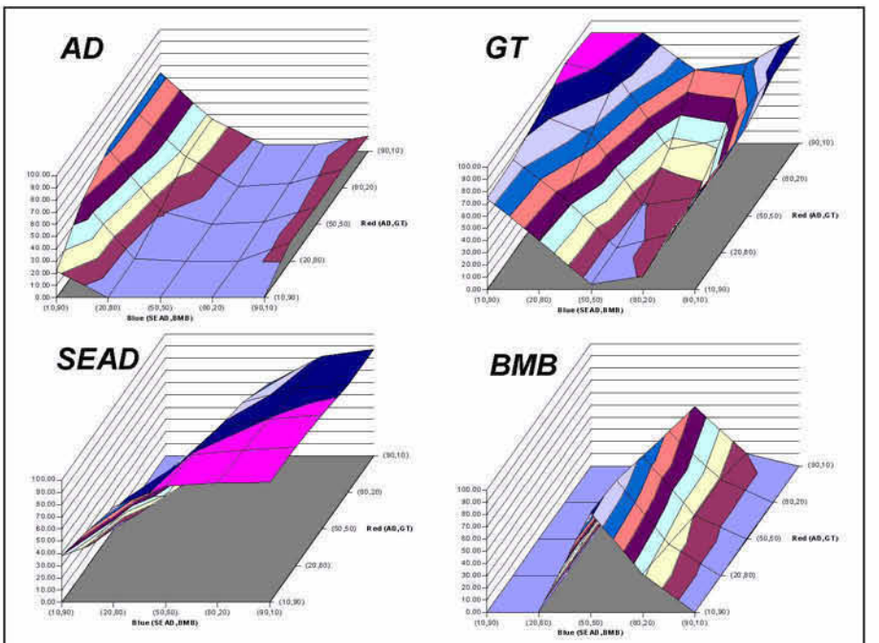


Fig. 3. Surviving % of each class of resource (EXP)

mechanisms and compare the resulting system behaviors with those of the full system. First, we performed a mean field abstraction (ABS3) that removes the effects of Blue strategy, spatial distribution, and the distinction among individual agents. Because this abstraction behaved differently from EXP, we examined two intermediate abstractions, one removing only blue strategy (ABS1), the other removing blue strategy and spatial distribution (ABS2). We present the abstractions in logical sequence rather than in chronological order. Due to space limitations, we discuss only those details that illustrate the impact of our successive abstractions.

4.1 ABS1: Ignoring Blue Strategy

Blue units find Red targets by climbing pheromone gradients. A logical abstraction is to "cut off their noses," moving Blue randomly rather than in response to pheromone signals. Figure 4 shows the strength of Red in Blue at the end of the game under these conditions. The landscape has the same general features as Figure 2.

We can compare the two by subtracting at each point the Red in

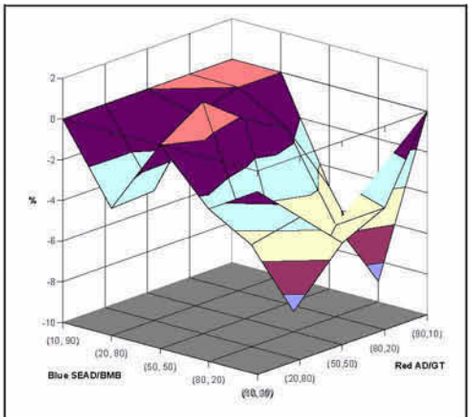


Fig. 4. BMB survival (EXP – ABS1)

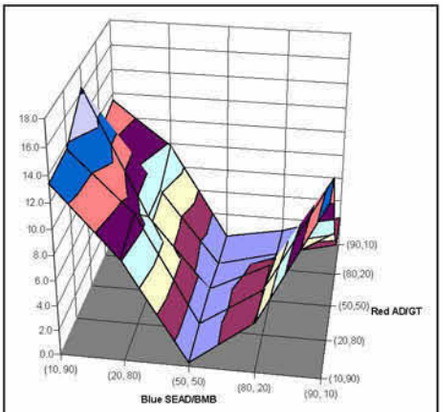


Fig. 5. Red Strength in Blue, without pheromone effects (ABS1)

Blue strength when Blue moves randomly from that when Blue follows pheromones, as in Figure 5. Because Blue seeks to keep Red out of Blue territory, differences less than 0 represent a net contribution of the Blue mechanisms. Figure 5 shows that our mechanisms are generally effective, with the greatest benefit at 10% SEAD, 50% AD. There are two exceptional regions where random wandering outperforms pheromones.

The first is when Red AD is above 50%. In this region, BMB survival is worse in EXP than in ABS1 (Figure 6), leading us to hypothesize two possible causes for the difference. It may result from the movement rule we have assigned to AD, to follow GT weighted by BMB pheromones. If BMB movement is regular (guided by slow-moving GT), AD's can position themselves more effectively. When BMB move randomly, AD's have a harder time positioning themselves for maximum impact. Another explanation recognizes that with high AD coverage around GT, a frontal attack of BMB on GT takes them into the most deadly opposition, while random movement will sometimes encounter weaker Red units that they can more effectively pick off. Distinguishing these alternative effects requires further experiments.

Lesson: Within the same problem domain, parametric differences can lead to a very different interaction between the agents and their environment. Designers of agents need to take these differences into account.

Second, even with low Red AD, SP mechanisms make little or no contribution at 50% SEAD, probably due to a lack of opportunity. The valley around 50% SEAD and low AD is so favorable to Blue that Blue's strategy makes little difference.

Lesson: Success may result from luck rather than intelligence. Environmental dynamics can be so strong that agent intelligence makes little or no difference.

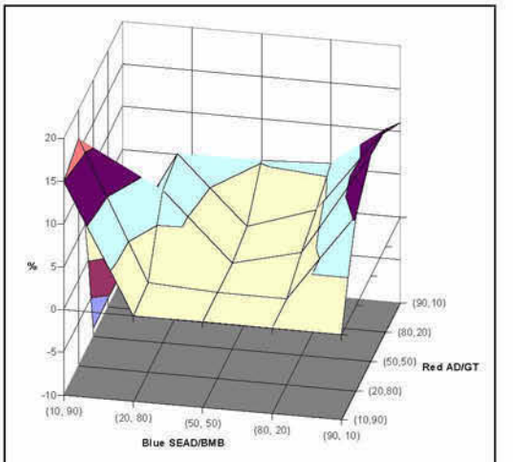


Fig. 6. AD survival (EXP – ABS1)

Figure 7 shows the difference in AD survival between EXP and ABS1. As we have seen, pheromones make little difference toward the low-AD end of the 50% SEAD valley, and help Blue at 10% SEAD, 50% AD. But they are a detriment around the edges of the valley. The right-hand ridge reflects the puzzle we have already seen in Figure 2: why should higher SEAD strength lead to better Red success? Figure 7 shows that this anomaly is reflected in AD survival. Initially, this circumstance is even more puzzling than high Red in Blue in this region. Why should higher SEAD help AD survival? By focusing our attention on the AD units, this plot leads us to the answer, a complex chain of interlocking events.

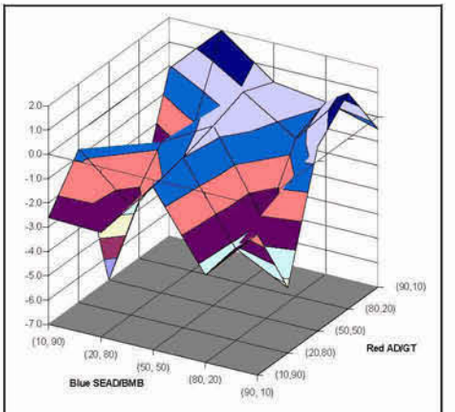


Fig. 7. Red in Blue (EXP – ABS1)

1. Rule 9 specifies that when BMB attacks GT, the losses on each side depend on the ratio GT/BMB. With high SEAD, a package contains fewer BMB's, and the GT/BMB ratio is higher. For very high SEAD levels, BMB is at a disadvantage in an encounter with GT, accounting for the lower survival of BMB at high SEAD.
2. AD's are attracted by GT pheromones, weighted by BMB pheromones. The BMB population falls off at high SEAD levels, both intrinsically and because of the dynamic in the previous point. Thus AD movement becomes more random, and AD's are less likely to be found in the close vicinity of GT.
3. SEAD's travel with BMB's in Blue packages, which are attracted by GT. As AD's wander more, they are less likely to be near GT's, thus less likely to encounter SEAD's, and their survival increases.
4. Meanwhile (returning to the right-hand flap in Figure 2), the decreased population of BMB's leaves GT free to invade Blue territory.

Lesson: Even simple rules interact in complex and unanticipated ways that designers must seek to understand through careful analysis.

4.2 ABS2: Ignoring Spatial Distribution

ABS1 shows us the effect of turning off Blue's pheromone mechanisms, but Red's deliberate movement is another layer that we must peel away from the system behavior to understand the impact of the outcome rules. One way to remove the effect of these mechanisms would be to randomize Red's movement as well as Blue's, but this would still leave a dependency on Red's initial spatial distribution. Alternatively, we can remove space entirely, so that all units occupy a single sector.

Historically, we analyzed ABS2 in order to validate the mean field approach of ABS3, which is intrinsically non-spatial, so ABS2 pursues the single-sector approach. This abstraction changes how Red and Blue agents encounter one another. In the

spatial model, agents interact when they find themselves in a common sector. As a result, agent movement (whether random or purposeful) induces a distribution on how many agents can be engaged at a given time step. For example, Blue in a sector with no Red forces can neither cause nor receive battle damage. Under such circumstances, an "attack" command is effectively a no-op. When we place all resources in the same sector, we need another way to model how many resources will be engaged. Thus we define the proportion of each type of unit that will execute each eligible command at each time step. In the results reported here, we assign the following parameters (parameter set a), based on the results from this same set in ABS3:

- AD: 0% Hide, 0% Deceive, 10% Fire, 90% Relocate (to the same sector)
- GT: 80% Hide, 20% Advance (to the same sector)
- SEAD: 10% Rest, 90% Attack AD, 0% New Sector
- BMB: 60% Rest, 20% Attack AD, 20% Attack GT , 0% New Sector

For example, at a given time step, a randomly selected 80% of the GT's will Hide, while the others will Advance (thus being vulnerable to attack).

Figure 8 summarizes some results from these parameters, compared with EXP and ABS1. These plots show several interesting features.

The topography in ABS2-a is shifted toward lower SEAD percentages, relative to that in EXP and ABS1. The valley in surviving GT and the peak in surviving BMB now fall between 20% and 50% SEAD, rather than at or beyond 50% SEAD as before. The location of the valley reflects the penalty imposed by Rule 5 when the ratio of SEAD to BMB falls below 1/2. In ABS1, SEAD and BMB are packaged based on the overall percentage of SEAD, which is thus involved in any combat. In ABS2-a, the proportion of SEAD and BMB in a conflict depends not only on the overall percentage of SEAD, but also on the number of each that is resting and out of action on a given cycle. The command percentages in Figure 8 make 90% of SEAD available to attack on any given cycle, but only 40% of BMB. Thus the effective SEAD percentage is more than twice the overall SEAD percentage, and ABS2-a shows the same effect at 20% SEAD that EXP and ABS1 show at 50% SEAD.

An obvious fix is to fit the command percentages more carefully to the distribution induced by movement in the spatially distributed case. Such a fit is more easily requested than delivered. The complexity of various movement rules makes an analytical derivation intractable. The desired distribution is probably not even stationary, since population changes over a run will change the probability that two units will encounter one another. One could use a visual or statistical match between performance landscapes such as those in Figure 8 to determine command percentages experimentally. More fundamentally, these observations call into question the validity of aspatial models of spatially distributed problems.

Lesson: Space is not just a neutral medium in which agents interact. It plays an active and complex role in their interactions, a role that is difficult if not impossible to capture without modeling space directly. Techniques that centralize distributed environmental processes through a single architectural module, such as a boundary interface [12], may inadvertently distort a system's behavior.

ABS2-a also differs from EXP and ABS1 in the nonlinearity of GT's dependence on AD percentage. In the previous experiments, the valley rises monotonically as AD increases. In ABS2-a, this rise peaks at 50% AD, then falls sharply for higher AD

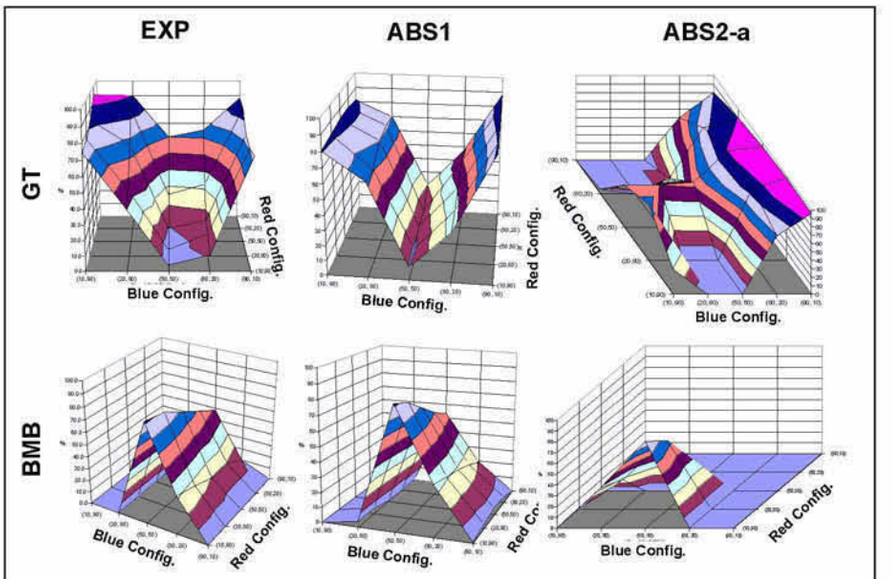


Fig. 8. Comparison of surviving GT and BMB percentages for EXP, ABS1, and ABS2-a

percentages. At this point, we do not have a detailed explanation for this feature. It is unlikely that we will devote considerable effort to understanding it, since the basic lesson of ABS2-a is that removing spatial distribution entirely from the model is not a fruitful approach to our objective of factoring agent effects from environment effects.

4.3 ABS3: Ignoring Unit Identity

Execution of agent-based models can be time-consuming (in our case, 7 minutes for 1200 hours). Eleven replications at each of 5x5 = 25 Red/Blue configurations require over 32 hours. A parallel equation-based model (ABS3) requires about 1.5 seconds to simulate 1200 hours. Significant differences between agent-based and equation-based models [18, 23] make the agent-based model the gold standard for evaluating our pheromone methods, but rapid surveys of parameter space with an equation-based model might guide slower verification using the agent-based model.

ABS3 uses a population-based modeling approach where the aggregated strength of all units of one type (AD, GT, SEAD, BMB) is represented in the size of one distinct population. The size of a population changes over time. The change is determined by the portion of each population that engages in combat in each discrete time step and by the losses inflicted in theses combats.

We represent the population dynamics in a set of difference equations that capture both the combat composition and the outcome rules. For example, in the GT population the population size is reduced by in every step by

$$\Delta GT = g(GT, Advance, BMB, AttackGT) * BMB * \frac{c(t, BMB, AttackGT)}{c(t, GT, Advance)}$$

where

- $g(X,a,Y,b)$ represents the percent losses of a group of units of type X that executes the command a when it engages a group of units of type Y that executes the command b at a time step. Losses are specified in outcome rules five to nine.
- $c(t,X,a)$ specifies the combat composition, and represents the percentage of population X that executes the command a at time t. The ABS3 experiments all assume a constant combat composition: $c(t1,X,a) = c(t2,X,a)$ for all pairs $(t1, t2)$.

ABS3 initializes the four populations to represent the initial strength of the combatants. Then, for a specified number of time steps, it

- computes the decrease in the population size for each population;
- limits the computed losses to the portion of each population that is actually engaged in combat (given by: $X*c(t,X,a)$, where a is an attack command); and
- applies the losses.

The number of time steps is the same as the number of calls in a comparable run of EXP to the function resolving combat situations.

ABS3 yields landscapes that match those from EXP and ABS1 qualitatively, but not in detail. The differences might be explained either by the move from an agent model to an equation one, or by the collapse of space. ABS2 teases those rival effects.

As we have seen, collapsing space does make a difference, due largely to the necessity to capture in static command probabilities the distribution of activities induced by agent encounters as they move through space. Comparison of ABS2 with ABS3 shows that the move from agents to equations has other effects as well.

Figure 9 compares three pairs of GT and BMB landscapes. ABS3-a uses parameter set a (defined in Section 4.2), and shows that even with a space-free model, command

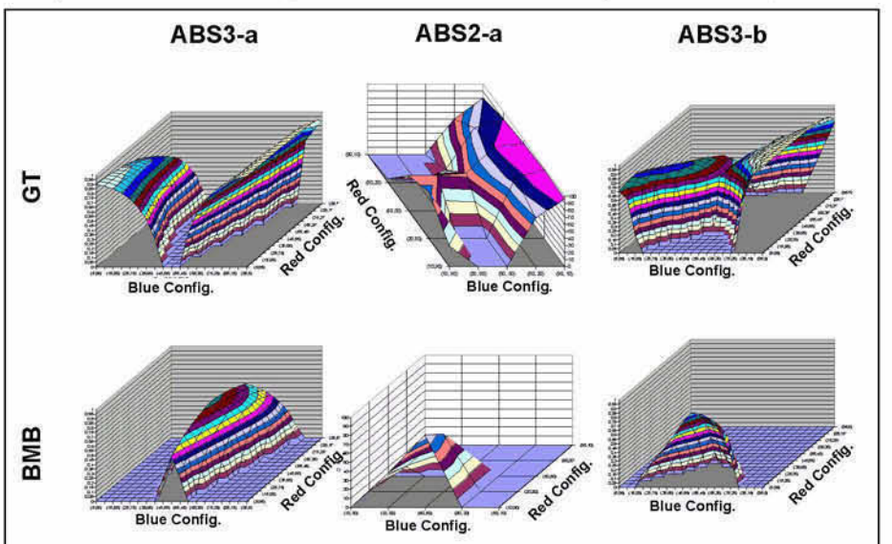


Fig. 9. Abstracting away from individual agents leads to further changes in behavior

percentages can be tuned to produce landscapes similar to those in ABS1 (or, for that matter, EXP; compare Figure 3). We ran ABS2-a with these same percentages to test whether the shift to an equation-based model makes a difference. Figure 9 shows that it does. The percentages that produce realistic landscapes in ABS3-a lead to the anomalies we have already discussed in ABS2-a. The third column in Figure 9 shows landscapes in ABS3-b, with different command parameters chosen to make these landscapes resemble ABS2 (AD = {0.0,0.0,0.5,0.5}; GT = {0.8,0.2}; SEAD = {0.45,0.55,0.0}; BMB = {0.2,0.4,0.4,0.0}).

Thus ABS3 can show us the existence of interesting non-trivial performance landscapes, but for a given set of parameters, it cannot reliably tell us either the location or the topology of their features. The salient difference between ABS2 and ABS3 is that ABS2 retains distinct agents, while ABS3 represents only the aggregate strength of the entire population of agents of a given type (thus, a single strength for each of AD, GT, SEAD, and BMB). The strength of individual agents in ABS2 evolves from the engagements in which each agent is involved, and thus summarizes that agent's history. ABS3 loses this history. The simulation logs show different evolution of the total strength over time in the two cases, leading to the different final outcomes reflected in Figure 9. The effect is closely related to the sensitive dependence of nonlinear systems on initial conditions. Once individual agents in ABS2 come to differ slightly in their strengths, their subsequent evolution can diverge greatly, leading to changes in the outcome of subsequent combats. ABS3 cannot track these different histories, and so is insensitive to their results.

Lesson: Like spatial distribution, ontological distribution (distributing processing over independent interacting processes) substantially and non-intuitively affects a model's results. Whether or not a mean-field model works in a given situation is an empirical question. Such models must be carefully validated against agent-based models before being trusted.

5 Discussion and Summary

The world is too complicated a place to understand as it exists. Science seeks to understand it by abstracting away details to leave a simplified system, and manipulating that system with a modeling technology (such as mathematical analysis or simulation). The validity of this process requires that neither the abstraction nor the modeling representation substantially change the behavior of the system. As our experience shows, both of these requirements are easily compromised.

First, designers of agent-based systems typically pay more attention to the agents than to the environment. The complex interactions discussed in this paper show that the environment deserves more attention. Our observations support researchers in embodied cognitive science [19] who argue that the agent and its environment must be designed together. The behavior of interest is that of the whole system, and only by considering the environment with the agent can we reliably design systems that do what we wish. The abstraction process exemplified in this paper is a methodological tool that can make us aware of how our systems interact with their environments.

Second, modeling technologies are not content-neutral. They can introduce artifacts determined more by the modeling technology than by the system being modeled. Earlier researchers have pointed out such effects within agent-based models, based on differences between synchronous and asynchronous execution [10, 13]. Our results in this paper reinforce our earlier observations [18] about the loss of ontological distribution in an equation-based model, a conclusion reinforced by Shnerb et al. [20].

System modeling and simulation are not impossible. In fact, they are unavoidable in engineering agent-based systems, due to the analytical intractability of typical systems and their strongly nonlinear behavior [15]. However, simulation is nontrivial. It forms a new way of doing science, alongside physical experimentation and mathematical analysis. These classical modes have evolved methodological guidelines for reliable results. Effective simulation science requires the development of similar guidelines, and the particular potential of agent-based modeling suggests that agent researchers should be in the forefront of developing this methodology.

Acknowledgements

Olga Gilmore and Murali Nandula of the ERIM CEC staff contributed significantly toward the experimentation reported in this paper. This work is supported in part by the DARPA JFACC program under contract F30602-99-C-0202 to ERIM CEC. The views and conclusions in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

6 References

1. Bonabeau, E., Dorigo, M., and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. New York, Oxford University Press, 1999.
2. Bonabeau, E., Henaux, F., Guérin, S., Snyers, D., Kuntz, P., and Theraulaz, G. Routing in Telecommunications Networks with "Smart" Ant-Like Agents. Santa Fe Institute, Santa Fe, NM, 1998.
3. Brueckner, S. *Return from the Ant: Synthetic Ecosystems for Manufacturing Control*. Thesis at Humboldt University Berlin, Department of Computer Science, 2000.
4. Cabri, G., Leonardi, L., and Zambonelli, F. Context-dependency in Internet-agent Coordination. In Omicini, A., Tolksdorf, R., and Zambonelli, F., Editors, *Engineering Societies in the Agents' World*, *Lecture Notes in AI*, Springer, Berlin, 2000.
5. Dorigo, M., Maniezzo, V., and Colorni, A. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(1):1-13, 1996.
6. Drogoul, A. *De la Simulation Multi-Agents à la Résolution Collective de Problèmes*. Thesis at University of Paris IV, 1993.

7. Drogoul, A. When Ants Play Chess (Or Can Strategies Emerge from Tactical Behaviors? In *Proceedings of Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '93)*, pages 13-27, Springer, 1995.
8. Ferber, J. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Harlow, UK, Addison Wesley Longman, 1999.
9. Goss, S., Aron, S., Deneubourg, J. L., and Pasteels, J. M. Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften*, 76:579-581, 1989.
10. Huberman, B. A. and Glance, N. S. Evolutionary Games and Computer Simulations. *Proceedings of the National Academy of Science USA*, 90(August):7716-7718, 1993.
11. Kott, A. SEADy Storm. 2000. Web Page, http://www.cgi.com/web2/govt/seadystorm.doc.
12. Moro, G. and Viroli, M. On Observing and Constraining Active Systems. In Omicini, A., Tolksdorf, R., and Zambonelli, F., Editors, *Engineering Societies in the Agents' World*, *Lecture Notes in AI*, Springer, Berlin, 2000.
13. Page, S. E. On Incentives and Updating in Agent Based Models. *Computational Economics*, 10:67-87, 1997.
14. Parunak, H. V. D. 'Go to the Ant': Engineering Principles from Natural Agent Systems. *Annals of Operations Research*, 75:69-101, 1997.
15. Parunak, H. V. D. From Chaos to Commerce: Practical Issues and Research Opportunities in the Nonlinear Dynamics of Decentralized Manufacturing Systems. In *Proceedings of Second International Workshop on Intelligent Manufacturing Systems*, pages k15-k25, K.U.Leuven, 1999.
16. Parunak, H. V. D. and Brueckner, S. Ant-Like Missionaries and Cannibals: Synthetic Pheromones for Distributed Motion Control. In *Proceedings of Fourth International Conference on Autonomous Agents (Agents 2000)*, 2000.
17. Parunak, H. V. D., Kindrick, J., and Irish, B. Material Handling: A Conservative Domain for Neural Connectivity and Propagation. In *Proceedings of Sixth National Conference on Artificial Intelligence*, pages 307-311, American Association for Artificial Intelligence, 1987.
18. Parunak, H. V. D., Savit, R., and Riolo, R. L. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. In *Proceedings of Workshop on Multi-agent systems and Agent-based Simulation (MABS'98)*, pages 10-25, Springer, 1998.
19. Pfeifer, R. and Scheier, C. *Understanding Intelligence*. Cambridge, MA, MIT Press, 1999.
20. Shnerb, N. M., Louzoun, Y., Bettelheim, E., and Solomon, S. The importance of being discrete: Life always wins on the surface. *Proc. Natl. Acad. Sci. USA*, 97(19 (September 12)):10322-10324, 2000.
21. Steels, L. Cooperation between Distributed Agents through Self-Organization. Vrije Universiteit Brussel AI Laboratory, 1989.
22. White, T. and Pagurek, B. Towards Multi-Swarm Problem Solving in Networks. In *Proceedings of Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 333-340, IEEE Computer Society, 1998.
23. Wilson, W. G. Resolving Discrepancies between Deterministic Population Models and Individual-Based Simulations. *American Naturalist*, 151(2):116-134, 1998.

On Observing and Constraining Active Systems

Gianluca Moro and Mirko Viroli

DEIS - Università di Bologna
Via Rasi e Spinelli, 176, 47023 Cesena (FC), Italy
{gmoro,mviroli}@deis.unibo.it

Abstract. While agents have emphasised the notion of active software components, they are not likely to be the only active components in agent-based systems. In this paper, we first discuss the general notion of active system, and show how it relates with the issue of the consistent observation of distributed and heterogeneous multi-component systems. Then, we introduce the concept of boundary interface as a methodological abstraction for the engineering of a society's environment composed by active systems, allowing observer agents to be provided with three different kinds of consistent environment views, featuring observable, controlled, and constrained consistency, respectively.

1 Introduction

Agent-based systems have emphasised the notion of active components, which do not simply react to invocations or queries, but exhibit instead some observable behaviour as emerging from their inner computational activity. Agents [30] are indeed such sorts of systems, but what is emerging from current experience with the engineering of complex software systems is that they are not likely to be the only kind of active entity populating the agent space.

This is easy to see when taking into account systems modelling highly dynamic domains, where some knowledge sources represent portions of the world whose state is changing in an unpredictable way. Agents in charge of monitoring or controlling such components are unlikely to be designed as polling observers, continuously checking knowledge sources to verify possible state changes. Instead, they are more easily to be thought of as sorts of registered observers, which have to be notified when something happens in the part of the world they are interested in. As a result, the general notion of an active system comes in, which includes not only agents, but also any component or set of components of an agent-based system that is able to manifest any change to its state as an observable event.

In this paper we focus on agent societies, that is we consider multi-agent systems with a global social task. In such a framework, agents typically have to take decisions and perform computation on the basis of the evolution of their local environment, which is composed not only by other agents, but also by knowledge sources. So in general, we call society's *environment* the set of all these knowledge sources, which we model as active components manifesting an observable

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 34–51, 2000.

behaviour through the notification of events. In general such an environment lacks a centralised control, has unpredictable dynamics, and is typically a distributed system of heterogeneous sub-components with no pre-existing cooperation. This framework highlights a key issue of both design and implementation: that of an agent's consistent observation.

Generally speaking, agent systems need often to deal with semantic relationships between different components which are often not easy to be implemented. This makes it difficult or even impossible to provide agent observers with a consistent view of the resulting system, in that the observation may require a higher abstraction level with respect to the one provided by single components.

Let us consider an example derived from a real application experience of a big hospital in Bologna [21]. A patient is subjected to several medical exams in different wards A, B, C, respectively managed by three different subsystems built over the time as independent parts without cooperative behaviours. An observer agent, representing for instance the patient's family doctor, is not interested in the single exam, carried out in the single ward, but only in the three exams as a whole, which represents for him/her a consistent view of the patient. But the three subsystems do not cooperate, so how can the problem be resolved without rebuilding all over again?

The first obvious solution is to charge the observer agent with the burden of consistency. However, this doesn't seem to be an effective and satisfactory approach: when heterogeneous subsystems are semantically correlated, it is likely to have many different observers asking for the same kind of consistent view, connecting sub-components at a higher level of abstraction. If observer agents directly accept the notifications sent by the systems, the computational overhead needed to obtain the consistent view would be replicated into each agent, causing problems of redundancy, maintainability and correctness. So, semantic correlations between structurally unrelated components are more likely to be faced outside the observers, by factorising consistency in some infrastructural abstraction.

To this end, the first goal of this paper is to introduce an abstraction, called boundary interface, which can be used as a methodological tool to build up observably consistent active systems from possible heterogeneous, independent, distributed, and decentralised subsystems, freeing observer agents from the charge of implementing consistent views. Taking the point of view of a society, the boundary interface may be seen as an infrastructure mediating between the society's environment and the agents with the task of monitoring it. The boundary interface provides for higher level, global and consistent views of the environment, thus freeing the agents from any burden of managing heterogeneity, synchronization and accommodation of the environment's manifestation.

We go on to discuss how such an abstraction can be differently instrumented, to provide for three different notions of consistency: observable, controlled, and constrained, at a growing level of expressing power. As a result, the notion of boundary interface subsumes the twofold role of providing observer agents with a consistent perception of complex observed systems, and possibly controlling their

evolution over time according to some super-imposed consistency constraints. In particular when the environment's behaviour has to be adapted in order to better fulfill the requirements imposed by the society's global task, then the boundary interface can be used to constrain and control its active sub-components, so as to adapt its evolution and free the agents from dealing with further local management.

The primary goal of this paper is then to highlight the need for taking into account the design of the environment of a society of agents. To this end, the boundary interface can be seen as a methodological abstraction providing for the design of those consistency rules modelling any kind of adaptation of the environment's behaviour and of its manifestation that may help in processing the social task.

The reminder of the paper is organised as follows. Section 2 describes the conceptual framework of the boundary interface, illustrating its features and its motivations. Section 3, 4 and 5 describe the logical architecture for the boundary interface when it materialises respectively an observable consistency, controlled consistency and a constrained consistency. Section 6 shows an example of monitoring, controlling and constraining of active systems modeling mobile robots, while Section 7 briefly outlines an architecture for a comprehensive boundary interface. To conclude, Section 7 reports the related works.

2 Conceptual Framework

When a system effectively represents portions of the world, the status evolution of the world coincides with the status evolution of the system. For instance, the status evolution of a database represents the status evolution of the portion of the world the database models. Systems are designed for serving the concurrent requests of proactive agents preserving their internal consistency (i.e. constraints and rules of the modeled information). In order for an observer to take decisions correctly, the observed system should be able to reveal only consistent views. Known models and technologies [24,1] allow a system to preserve and supply consistent views of its status only when dealing with client-server like query requests.

However, in order to discover possible changes, an observer should not have to work on a polling basis, continuously querying the system, because of the following reasons:

- *lack of effectiveness* - this approach does not guarantee the perception of all status changes over the time, in fact the system could change more rapidly with respect to the query frequency;
- *lack of efficiency* - the system is forced to serve all the queries even when nothing is changed, thus wasting computational resources.

To overcome these problems, observers should be notified by the system itself only when something they are interested in happens. To this end, system components should be designed as active systems, explicitly manifesting their status

changes. Each observer interested in some portion of the system should register itself to one or more sub-components, and then be notified when any of these sub-components changes. This is a very common interaction pattern typically known as the publish/subscribe protocol. Its importance in the structuring of active systems has been highlighted in many different areas: active databases [1,12], object frameworks [11,24] and coordination models [2,31,16].

In this way however, each observer agent should also carry out the task of rebuilding consistent views of the system's states of interest directly from sub-component events. Indeed, this task often requires some low-level knowledge on the system and on the update operations involving those components. So, it is likely to be implemented and executed in a redundant way by each observer, with risk of errors and waste of computational resources. Moreover, the resulting system would be unmanageable, since extensions or modifications would be replicated in each observer agent.

Our proposal is to factorise the process of building and notifying consistent views of a multi-component system in an abstraction we call *boundary interface*. This software layer is a sort of output interface complementary to the traditional usage interface (or rather entry interface). Its main goal is to publish a set of higher level events grouping system events in macro-events, each representing an atomically notifiable and consistent view, of the multi-component system.

In general, the observation can involve two or more systems or subsystems possibly independent and without cooperative behaviours. In this case the boundary interface could realise a higher level system, resulting from the combination of independent sub-systems, by super-imposing higher level consistency rules. Let us consider the hospital example mentioned above. The higher level consistent view of the patient, required by the family doctor, does not belong to any of the three subsystems, but it could be built by the boundary interface. This will be mainly realised by collecting and processing the events notified by all the single subsystems.

When the boundary interface builds consistent views of a multi-component system, we say that it realises the notion of *observable consistency*. The boundary interface, in this case, allows registered observers to perceive consistent manifestation of status changes of the system $\mathcal{S}$, intending $\mathcal{S}$ as a multi-component system modelling the agent society's environment. The only requirement for $\mathcal{S}$ is the capability of issuing the status changes of its parts (i.e., its components or subsystems).

With this framework the boundary interface can be easily though as a software layer that not only monitors $\mathcal{S}$ and notifies registered observers, but that can also control in some way the evolution of $\mathcal{S}$. In this case, the system $\mathcal{S}$ should be able to issue the proposals of the status changes of its parts before they are actually performed. The boundary interface may then decide to refuse or accept these proposals by applying/using new super-imposed notions of consistency. If and only if these proposals are accepted then the corresponding status change becomes effective.

For instance, let us consider two independent industrial robots consisting of two mobile arms. Each one proposes its moving intention to the boundary interface, which decides to refuse or accept them according to super-imposed rules. These rules may be designed, for example, to avoid collisions or dangerous movements. Notice that this notion of consistency cannot belong to any of the two robots, but is more likely to be viewed as belonging to the overall system, made up of the two robots as a whole. When dealing with this new semantics, we say that the boundary interface implements a notion of *controlled consistency*.

Another task that we may need the boundary interface to implement is the constraining of the evolution of the system. In this case the requirement for $\mathcal{S}$ is to both issue status changes of its parts and to accept status change requests from the outside. In this scenario, the boundary interface may act as a constrainer, since it can monitor the evolution of the system and decide to change its evolution by sending some change request. Namely, the boundary interface constrains the evolution of $\mathcal{S}$ by pushing it towards a specific direction (intended as status) according to a super-imposed consistency.

Considering the previous example of the two robots, a super-imposed consistency could establish that when the robot *R1* is late on some job with respect to *R2*, the boundary interface may constrain *R2* to slow down in order to be aligned with *R1*. In this case, we say that the boundary interface realises a notion of *constrained consistency*.

3 Observable Consistency

Consider again the system $\mathcal{S}$ modelling the society's environment, which is composed of different, possibly distributed and heterogeneous sub-components, each able to notify the events of change on its status or on a part of it.

When we talk about "events" we explicitly refer to the publish/subscribe protocol. In particular we consider the case in which a publisher, the active system, fires events to one or more subscribers, the observers. Each publisher exports one or more types of event, and a subscriber may register for one of them by sending a specific request message. So the publisher, when some condition is satisfied or an action occurs, may decide to fire one event to all the subscribers which have registered for it. This event consists in a message typically carrying information about the publisher (source of the event), the cause of the event, and some data value. In the remainder of the paper we will focus on those active systems which are able to manifest their activity using the publish/subscribe protocol, i.e., that are able to behave as publishers.

We call *system events* the change events that an active system $\mathcal{S}$ may fire. *Observers* are entities of the domain that can interact with $\mathcal{S}$ and are interested in its changes. They should behave as subscribers in the publish/subscribe protocol.

In general, the events the observers are interested in, which we call *output events*, belong to a different abstraction level than that of system events, which just represent changes on subparts of $\mathcal{S}$. So, the boundary interface is in charge

of allowing observer agents to perceive system changes by filtering and processing system events and producing output events. From the point of view of the publish/subscribe protocol, the boundary interface can be seen as a subscriber for system events and as a publisher for output events, while observers will behave as subscribers for output events. In this specific case the boundary interface will be called *observation interface*, and we say that it materialises an *observable consistency*.

Notice that in our framework we are not interested much in having the boundary interface dynamically registering for events, but we suppose that at startup-time it is instantiated once for all and connected in a static way to the corresponding active systems. The tasks that the observation interface implements are: (i) to correlate system events using some super-imposed policy, (ii) to group them and perform some computation over the data they carry, (iii) to pack output events, and (iv) to fire them to the interested observers. The logical architecture of the observation interface is shown in Figure 1.

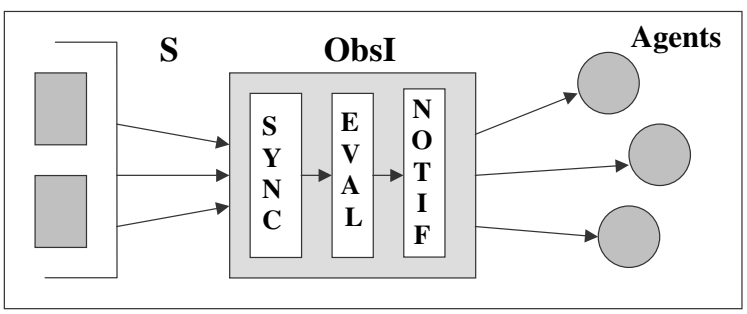


Fig. 1. Logical Architecture for the Observation Interface

When computing the data values carried by multiple events, a preliminary and fundamental operation, which we call *correlation*, is always requested. By the term *correlation* we mean the identification of a relation or a property over the events, whose satisfaction causes the data values they carry to be grouped and computed as a whole. In our architecture the correlation rules will be typically designed in order to identify the common cause for the generation of some system events, leading to the firing of one or more output events.

The part of the observation interface which realises this job is called Synchroniser. It takes all the system events that have to be processed, it classifies them, and it decides when to group some of them in order to fire a *grouped event* to the next logical sub-components. A grouped event is an event carrying the data values of a set of (correlated) system events; we use it in order to notify multiple data values into a whole. The strategies of synchronisation (also called correlation strategies) may be different, depending on the kind of system we are

observing and on the kind of elaboration that has to be performed. Some of them are:

- *Temporal correlation*, events coming at close times are grouped together, and the firing of a grouped event is performed when an internal clock sends its tic.
- *Size correlation*, events are put into a buffer and sent when the buffer becomes full.
- *Data correlation*, events carry some information that is used both to classify them and to decide when to send a grouped event.

When a grouped event is fired it reaches the Evaluator logical component. Its task is to produce the actual output event that will be notified to the observers. This is typically realised by performing some functional transformation of the data values the grouped event carries, and producing as a result an output event.

The final logical component of the observation interface is the Notifier. Its task is to keep track of the observers' registration, to accept output events from the Evaluator, and to decide which observer has to be notified. It exports a statically-fixed set of output events, representing consistent changes on logical sub-parts of $\mathcal{S}$, and observers should specify those in which they are interested. So the Notifier has to classify the events the Evaluator sends and decide which observer to notify.

When the output events are logically disjoined the implementation of the Notifier is quite simple. More likely there can be some causality relation between them, leading to implementations that provide the notification of all the observers which have registered for an output event that has some relation with the one that is occurring.

Consider in fact the case in which the observation interface monitors two subsystems $\mathcal{A}$ and $\mathcal{B}$ and exports three events respectively representing: changes on $\mathcal{A}$, changes on $\mathcal{B}$, changes on either $\mathcal{A}$ or $\mathcal{B}$. When an event of the third kind occurs, the observers for the first one and the observers for the second one might be notified too.

Another interesting issue is the way a Notifier should deal with multiple registrations, i.e., those cases in which a single observer registers for more than one output event. Suppose an observer is interested in receiving both the changes on $\mathcal{A}$ and $\mathcal{B}$, and registers for both the corresponding output events. Suppose then, that due to a message coming from the Evaluator the Notifier is willing to notify both the output events $\mathcal{A}$ and $\mathcal{B}$. If the observer receives both the notifications it would have still the problem of consistency, with the need of a further step of synchronisation. Given a single message coming from the Evaluator, a feasible notification policy should provide each observer for the notification of just one message, possibly containing the information about more than one output event occurred.

4 Controlled Consistency

When the only task of the boundary interface is to allow observers to perceive the system changes, then the only requirement for the environment $\mathcal{S}$ is to be able to fire events representing internal changes.

Another interesting task for the boundary interface is to control the evolution of $\mathcal{S}$ by observing its events and replying to each of them with an acceptance or a refusal feedback message. In this case, the requirement of the environment $\mathcal{S}$ is not just to be able to notify changes, but to propose changes waiting for their acceptance before the actual committing. When the boundary interface can handle this event's semantics we call it a *control interface*, and we say that it materialises a *controlled consistency*.

The control interface super-imposes some (consistency) rules that are used to control the environment's evolution. The corresponding semantics is somewhat similar to that of the interaction between a constrained property and the corresponding vetoer in the JB (Java Beans [15]) architecture. It can be considered a direct extension of the publish/subscribe protocol, where the publisher waits for a feedback from the subscribers each time an events is fired. Examples of systems that can handle this behaviour are the transactional ones. They can be used so that if an event handler routine fails (refuse message from the observer) the transaction that caused the event to be fired fails too, and is rolled back. This is in fact the typical usage of a vetoer object in an EJB (Enterprise Java Beans [11]) framework. Other examples may include non-transactional systems composed by agents sending the notification and simply waiting for the response, possibly carrying on a parallel activity.

Notice that in order to avoid any problem of autonomy of control, a system should export "controllable" events only if it is actually able to deal with refusal feedbacks. In fact an event will be considered actually occurred by the control interface only if it has been accepted.

A logical architecture for this kind of boundary interface may be the one shown in Figure 2. The first logical component of the interface, as for the observation interface, is the Synchroniser. In fact, we already argued that the operation of correlation is always necessary when computing data coming from different events. The Synchroniser will be realised as previously discussed for the observation interface: it collects system events using some local correlation policy, and then fires grouped events to the next component.

In this case the grouped event reaches the Checker, which stores the rules used to decide if each single system event should be rejected or accepted. When this decision has been taken, it is communicated to the system through a feedback message.

Moreover in this architecture, one may think of allowing the decision of the Checker to be observed. The system events that have been accepted may be in fact notified to interested observers for further processing. We call the events a control interface may fire *observable events*. As we will see in Section 7, these events may be handled as system events by a subsequent interface, allowing for instance to compose a control interface with an observation interface.

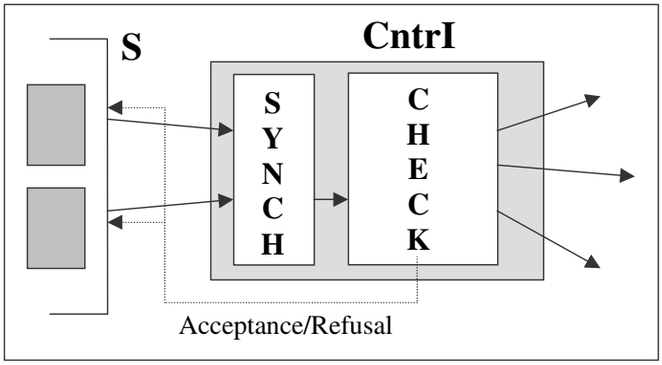


Fig. 2. Logical Architecture for the Control Interface

5 Constrained Consistency

A more powerful way of interacting with the evolution of a system, than by just controlling it through the acceptance or the refusal of its proposal of changes, is to constrain its processing by querying new updates. So, another interesting pattern of interaction between an environment $\mathcal{S}$ and its boundary interface is the one in which the latter acts as a constrainer for the former.

To this end, $\mathcal{S}$ should be able to ask for completion proposals. It notifies internal changes and then accepts and executes corresponding external requests for new changes. Suppose $\mathcal{S}$ publishes n events to the interface, and that at a given time it notifies changes only through a subset of them. The boundary interface therefore, may try to complete the information sent by $\mathcal{S}$ by calculating acceptable/required values for the remaining events, and notifying them to the system through feedback messages, which we call *completion events*. Then $\mathcal{S}$ may then try to perform the changes requested, possibly replying with a refusal message in case of fail.

When dealing with this semantics, we call the boundary interface a *constraint interface*, and we say that it materialises a *constrained consistency*. In Figure 3, the logical architecture for this kind of boundary interface is shown.

Its structure is very similar to that of the control interface, in that both interact with the environment in order to monitor and then adjust its evolution. The main difference is that in the former case the environment proposes changes, while in the latter it asks for a completion on the information it has.

As usual such a completion is implemented in two steps: one where events are correlated and grouped together (synchronisation task), and another where some elaboration is done over the values carried by system events. When the completion has been determined, the Completer component notifies $\mathcal{S}$ sending the new data. Here again, in order to preserve the autonomy of control, $\mathcal{S}$ should issue events to a constrained interface only if it is able to accept (and possibly refuse) requests of changes corresponding to each of them.

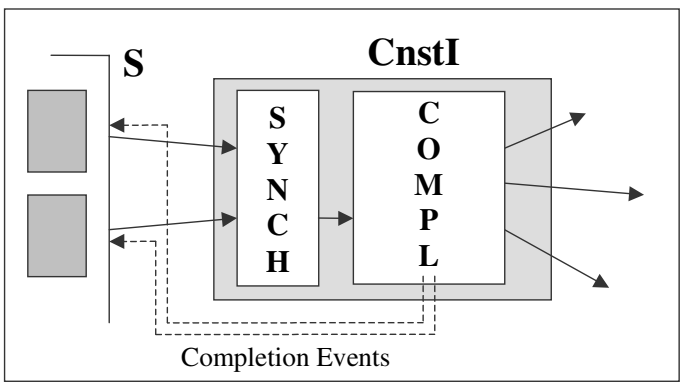


Fig. 3. Logical Architecture for the Constraint Interface

In case of a transactional system $\mathcal{S}$, the processing caused by the completion events should be executed within the transaction that fired the corresponding system events. This can be achieved if the system works using the two-phase commit protocol [1] and the constraint interface is a participant of the transaction. In an agent-based system one has to explicitly design an appropriate protocol of interaction in order to support the semantic of requested completion.

Analogously to the control interface furthermore, the constraint interface may let some events, which we call again observable events, to be notified to interested observers. Considering a completion request, observable events are fired if and only if $\mathcal{S}$ accepts all the completion events. In this case both system events and completion events are fired, since they represent the appropriate view of the constrained system. Obviously $\mathcal{S}$ should treat the completion events' requests so that they won't cause new proposals for completion again. This in order to avoid unexpected recursive behaviours.

6 An Example of Application

In previous sections we saw three different patterns of interaction between an environment system S and the logical infrastructure called boundary interface. In each case we analysed the requirements for the environment and an architecture for the boundary interface, which we renamed respectively observation interface, control interface, and constraint interface. In this section we show a brief example in which these three kinds of boundary interface can be exploited.

Suppose our environment system $\mathcal{S}$ is composed by three mobile robots, namely $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$, moving into a given area. Some observer agents are interested in monitoring information like the position of each robot, the mutual distances, which robots are going to collide, and so on. We suppose then, that all observers need to be notified at a given frequency, say each time per second, and that each robot can be viewed as an active system notifying changes on its position with unpredictable dynamics.

An infrastructure to solve this problem may exploit an observation interface taking the three system events from $\mathcal{S}$, namely `pA`, `pB` and `pC` (the position (x,y) of each robot) and exporting 9 output events: the three positions `oA`, `oB` and `oC`, the three mutual distances `dAB`, `dBC` and `dCA`, and three boolean values stating when a given robot is near enough to another one or to an obstacle, say `bA`, `bB` and `bC`. See Figure 4.

The Synchroniser includes an internal clock firing a tick each time a second. This tick causes a grouped event to be sent, carrying the oldest position received from each robot since the last tick occurred. The Evaluator computes the 9 values required for each output event, possibly using cached values for those positions that has not changed since the last time a grouped event arrived. The Notifier notifies each observer with those values that it has registered to, and that has been updated since the last time. For example, if since the last Synchroniser's tick, only robot $\mathcal{A}$ has changed its position, then only output events `oA`, `dAB`, `dCA`, `bA`, `bB` and `bC` will be sent.

Now suppose each robot does not notify changes on its position, but proposes new ones, which should be considered carried out only in case of acceptance. In this case we may think of using a control interface in order to avoid the distance between any two robots to be smaller than a give parameter. See Figure 5.

Having not to directly deal with observers, the Synchroniser of this interface may behave differently to that of the previous case, for example it can fire a grouped event each time a system event occurs, that is considering all the update proposals not to be correlated in any way. The Checker computes the distance between the three robots using the new value for the one that is willing to move and the oldest ones for the others, sending a refusal if any of the three distances is too small.

The system composed by the three robots ($\mathcal{S}$) and this control interface can be considered a controlled system, which is able to manifest its evolution through three system events (which are actually observable events of the control interface, according to the architecture explained in section 4). Hence the observer agents monitoring these system events perceives the evolution of a "collision-safe" robot system.

Finally, suppose each robot both notifies proposals for moves and accepts new positions to move to. In this case a constraint interface can be applied to the system in order to force the robots to stay sufficiently near to each other. So when a robot, say $\mathcal{A}$, decides to explore a new area, the others can be constrained to follow it. See Figure 6.

This can be realised for example with the same Synchroniser of the previous control interface, and with a Completer constraining the robots that are not moving, in this case $\mathcal{B}$ and $\mathcal{C}$, to follow $\mathcal{A}$, i.e., to move to a position near $\mathcal{A}$.

7 A Global Architecture for the Boundary Interface

Since by now we defined boundary interfaces implementing just one kind of consistency rule, in this section we describe an infrastructure dealing with all

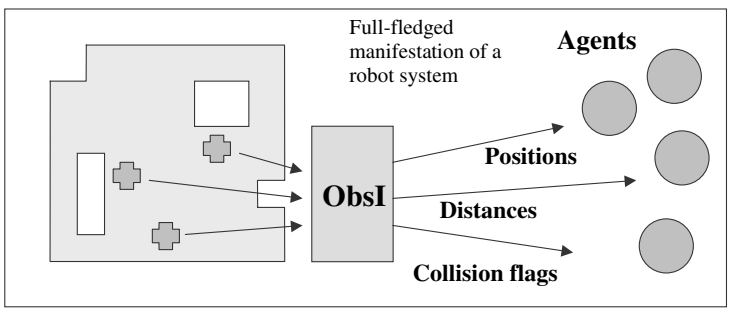


Fig. 4. Using an Observation Interface to monitor mobile robots

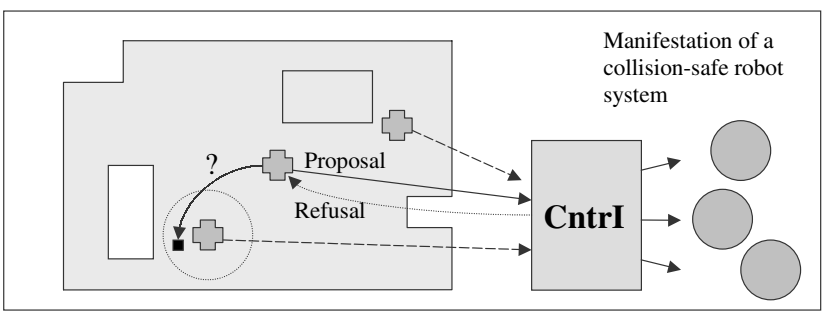


Fig. 5. Using a Control Interface to avoid robot collisions

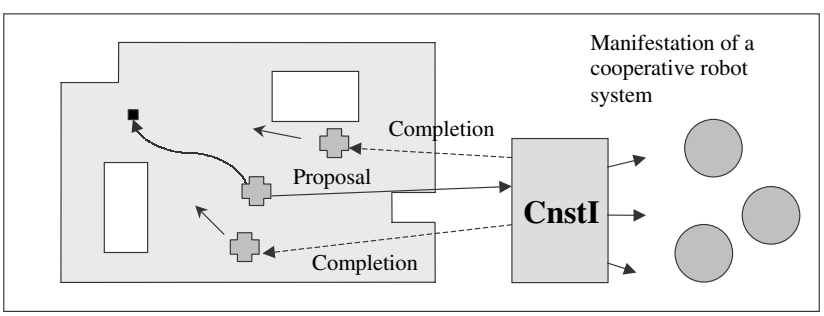


Fig. 6. Using a Constraint Interface to adapt the evolution of the robots system

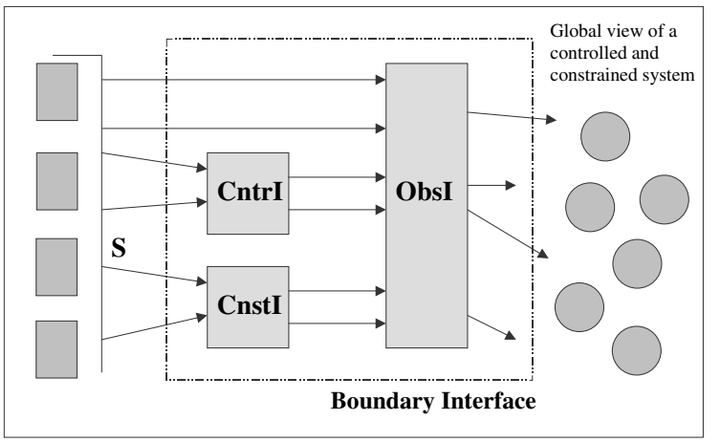


Fig. 7. Global logical architecture for the Boundary Interface

the three kinds of management, in a comprehensive way. So a possible integration of the observation interface, the control interface and the constraint interface is depicted in Figure 7.

By exploiting the observable events supplied by control and constraint interfaces, one may think of using an observation interface to monitor the evolution of the controlled and constrained parts of an environment. In this way, once the environment's behaviour has been adapted, observer agents may perceive its global and consistent view.

This is however the only kind of integration we can provide for. In fact the output events sent by control and constraint interfaces just represent notification of changes, hence they are only observable - they are not controllable nor constrainable. In general, for instance, we may think of making a control interface producing events representing proposals for evolution or requests for completion. However this would lead to a too much complicate architecture, where acceptance/refusal messages and completion proposals may be sent back to the environment through a set of cascading components. We believe that our choice for the architecture is the one that better allows to both keep simplicity and power.

The framework discussed in this paper is meant to be just a logical architecture. How a boundary interface could be actually implemented so as to fulfill requirements such as high scalability, high filtering speed, and the like, remains an open issue. In fact, the trivial implementation exploiting a single and centralised software component, gathering the enviroment's events and sending output events to all the agents, would be too much a bottleneck for the performance of the overall multi-agent system.

However, the problem of implementing certain tasks of the boundary interface in a centralised way is somewhat unavoidable, and furthermore it could also lead to some performance optimization.

Consider the example of medical laboratory described in Section 2. If more than one agent, say we have n of them, need to perceive consistent views of a patient, some part of which are supplied by m distributed laboratories, we may rely on two possible solutions: (i) we can send the events of all the laboratories to all the agents, leading to $n * m$ notifications, (ii) or we can provide a centralised agent gathering them, computing the consistent view, and sending output events to the agents, leading to just $n + m$ messages. So in general, when we need to build consistent views of semantically correlated notifications, it would be better to delegate the task once for all to a single component. Clearly, it would be also interesting to distribute these correlation tasks as much as possible, for instance by delegating the management of each consistency rule to a different centralised computational entity.

In particular we may rely on an agent-based architecture. So a feasible implementation for the boundary interface may exploit *source agents* that directly monitor active systems, each providing a wrapper for one of them, *synchronizing agents* that may be in charge of synchronizing notifications, *computing agents* that can be devoted to compute data of a higher level of abstraction, and finally *notification agents*, that supplies output events.

8 Related Works and Conclusions

Typically, a system is thought and designed only by facing the point of view of its use, that is, by providing for a set of functionality that some client may use to manipulate it and possibly to change its status. For this class of clients effective models and methodologies which guarantee that the system transits always across consistent status have been proposed, like Transaction Processing Systems [1,18].

In this paper, we focus on observers, which are mainly interested in being notified by the system when its status (or part of it) changes. Of course, in this case also, it is necessary that the observers are notified of consistent status changes of the system. However, the problem of determining models and frameworks for allowing consistent observations has not received the same attention. Currently, relevant research areas like object oriented systems, DBMS and agent technology seem to offer only implementation mechanisms for the observation. In fact, system's observers typically just accept events from reactive elements of the system. For instance, in object oriented systems, active entities are built over frameworks (like Enterprise Java Beans [11] and Corba [24]) exploiting design patterns such as Gamma's observer [13], and Schmidt's reactor and active object [26], which don't face the problem of consistency. In previous works [22,23] we proposed a solution applyable to object oriented applications, featuring an architectural pattern based on the observation interface.

In commercial DBMSs also (Oracle [25], Sybase [27],etc.) the problem of making the database active is solved at a mechanism level. In order to allow the notification of changes to observers, the database architecture has been modified by adding triggers [17,20]. Triggers are pieces of code attached to database tables

that can notify to a given port events of insertion, deletion and update of tuples (see for example Oracle Triggers [25]).

Some research prototypes of active DBMS have been developed on the basis of Event-Condition-Action model (ECA) and active rules model [1,12,10]. Some examples are HiPAC [9], Starburst [29] and Chimera [6]. Most of them support composite events [7] allowing internal ECA rules to be activated when two or more events occur. Our Synchroniser component performs a task that could be analogous to a composite event detection, but which is implemented outside the DBMS. This allows existing databases with minimal active capabilities to be turned into full-featured observable systems thanks to the observation interface.

Some coordination models [31,2] make use of publish/subscribe capabilities as well. JavaSpaces [16] for example deals both with event notification and transactional consistency. Its tuple spaces can be programmed so as to notify registered observers when a tuple matching a given template is inserted. A request for notification can specify whether the events should be fired directly when the tuple is inserted or waiting for the corresponding transaction to commit. Supposing a system's change is represented with a tuple inserted in a JavaSpace, and that a committed change can be considered consistent, this management allows for consistent changes to be notified to registered observers. With our framework a JavaSpace can be seen as an active system notifying its changes, hence we can apply an observation interface in order to add a super-imposed observable consistency, hence monitoring a co-ordinated system's evolution. It is not clear in which way a JavaSpace could be modified or extended in order to satisfy the requirements needed for being controlled and constrained.

The idea of taking into account the environment dynamics when designing a multi-agent system is supported by other works. [28] presents the analysis of a multi-agent system for the simulation of air resources defending a region from enemy attacks. This paper highlights the importance of designing together agents and their environment when multi-agent systems are used to simulate the real world.

The architecture analised in [8] shows the need for context-dependency when building an internet-agents architecture. In particular it proposes that the coordination rules for internet agents should be differently programmed in each execution environment so as to lead to different environment's behaviours. Moreover the agents can change the environment's programming in order to obtain application-dependent behaviour.

However, in general the society's environment is not yet considered as a first-class entity in multi-agent systems, both at the analysis and design level. Methodologies and tools for specifying and design agent-systems, such as Agent-UML [4] and the formal framework proposed in [14], often focuses on just agents, their behaviour and their mutual interactions. These frameworks seems to suggest to wrap the environment within one or more agents abstractions, but it is not already clear if this solution is good enough to model real systems. Other works related to our own, are [3] and [19]. In the former the idea of agents as active components is studied with greater detail, and the latter addresses

the design of event-data processing pipelines exploiting event-based multi-agent systems.

Finally we want also to mention [5], which is one of the very first papers addressing the idea of social order, that is investigating the structure, the interactions and the dependencies between agents building up a society with a peculiar social goal. Our paper addressed only one aspect of this idea, that is it provides for a methodology that can be used to adapt the environment's behaviour and manifestation according to a global social task.

To our knowledge, the problem of consistent observation has not yet been addressed in a systematic way. However, given that agent are likely to act in software system more and more on behalf of human beings, to sense the world and act on it consistently and timely by their own initiative, this problem is going to become more relevant for multi-agent systems than for any other kind of system. That is why we argue that the problem tackled by this paper, even though of general interest, seems to be even more important for agent-based systems.

References

1. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone, Database Systems Concepts, Languages and Architectures. McGrawHill, pag. 441-461, ISBN 007 709500 6, 1999 36, 37, 43, 47, 48
2. F. Arbab, I. Herman, and P. Spilling, An Overview of Manifold and its Implementation, *Concurrency: Practice and Experience*, 5:1, pages 23-70, February 1993. 37, 48
3. M. Amor, M. Pinto, L. Fuentes and J. M. Troya, Combining Software Components and Mobile Agents. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 55-68, held at ECAI 2000, Aug. 21-25, Berlin. 48
4. F. Bergenti and A. Poggi, Exploiting UML in the Design of Multi-Agent Systems. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 96-103, held at ECAI 2000, Aug. 21-25, Berlin. 48
5. C. Castelfranchi, Engineering Social Order. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 140-151, held at ECAI 2000, Aug. 21-25, Berlin. 49
6. Ceri, S., Fratemali, P., Paraboschi, S., and Branca, L. (1996). Active Rule Management in Chimera. *In Active Database Systems - Triggers and Rules For Advanced Database Processing*, pages 151–176. Morgan Kaufmann. 48
7. Chakravarthy, S., Krishnaprasad, V., Anwar, E., and Kim, S. K. (1994). Composite Events for Active Databases: Semantics Contexts and Detection. *In Proceedings of the 20th International Conference on Very Large Data Bases*, pages 606–617. 48
8. G. Cabri, L. Leonardi, and F. Zambonelli, Context-Dependency in Internet-agent Coordination. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 104-114, held at ECAI 2000, Aug. 21-25, Berlin. 48
9. U. Dayal, A. Buchmann, and S. Chakravarthy, The HiPAC Project, *In Active Database Systems (Eds. J. Widom and S. Ceri)*, pp. 177-206, MK, 1996. 48
10. U. Dayal, E. N. Hanson, J. Widom, Active Database Systems, Addison Wesley, September 1994. 48

11. Enterprise Java Beans Specification, Sun Mycrosystems, 1999, http://www.javasoft.com/products/ejb/docs.html 37, 41, 47
12. P. Fraternali, L. Tanca, A Structured Approach for the Definition of the Semantics of Active Databases, ACM Transaction on Database Systems, Vol. 20, No. 4, pp. 414-471, December 1995. 37, 48
13. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Mass., 1995. 47
14. V. Hailer, A. Koukam, P. Gruer, and J. Mueller, Formal Specification and Prototyping of Multi-Agent Systems. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 69-81, held at ECAI 2000, Aug. 21-25, Berlin. 48
15. Java Beans Tutorial, Sun Mycrosystems, Sept. 1997, http://java.sun.com/beans/docs/Tutorial-Sep97.ps 41
16. JavaSpaces Specification, Sun Mycrosystems, Jan. 1999, http://www.sun.com/jini/specs/js.pdf 37, 48
17. S. Kim, S. Chakravarthy, A Pratical Approach to Static Analysis and Execution of Rules in Active Databases, ACM, Las Vegas Nevada USA, pag. 161-168, 1997. 47
18. Sherri Kennamer, Microsoftcorn: A High-Scale Data Management and Transaction Processing Solution, SIGMOD'98 Seattle. WA, USA, 1998 ACM pag. 539-540 47
19. C. Koch and P. Petta, Multi-Agent Coordination of Distributed Event Data Processing. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 115-126, held at ECAI 2000, Aug. 21-25, Berlin. 48
20. G. Kappel, W. Retschitzegger, The TriGS Active Object-Oriented Database System - An Overview, Technical Report of Department of Information System, University of Linz, 1998. 47
21. G. Moro, A. Natali, M. Viroli, An Interactive Computational Model for Monitoring Systems, Tech-report. DEIS-LIA-006-99 n. 40, DEIS - University of Bologna 35
22. G. Moro, A. Natali, M. Viroli, An Architectural Pattern for Consistent Observation of Active Systems. *In Workshop Readers of the European Conference on Object Oriented Programming, 2000, Lecture Notes in Computer Science*, Cannes 2000. 47
23. G. Moro, A. Natali, M. Viroli,On the Consistent Observation of Active Systems. *Proceedings of the AI*IA/TABOO Joint Workshop 'Dagli oggetti agli agenti: tendenze evolutive dei sistemi software' (WOA 2000)*, Parma, May 2000. 47
24. Object Management Group, CORBAservices: Common Object Services Specification. Revised 09/12/98. 36, 37, 47
25. Oracle Corporation, PL/SQL User's Guide and Reference Release 2.3, Part No. A32542-1, chapter 8, February 1996 47, 48
26. Schmidt, D. Using design patterns to develop reusable object-oriented communications software. Commun. ACM 38, 10 (Oct. 1995),65-75. 47
27. Sybase. Sybase SQL Reference Manual: Volume 1. Sybase, Inc., 1996. 47
28. H. Van Dyke Parunak, S. Brueckner, J. Sauter, and R. S. Matthews, Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies. In proc. of the Workshop *Engineering Societies in the Agents' World 2000*, pages 1-14, held at ECAI 2000, Aug. 21-25, Berlin. 48
29. Widom, J. (1996). The Starburst Active Database Rule System. *IEEE Transactions on Knowledge and Data Science*, 8(4): 583–595. 48
30. Wooldridge, M. J., Jennings, N. R., Intelligent Agents: Theory and Practice, The Knowledge Engineering Review 10:2, pp. 115-152. Cambridge University Press, 1995. 34

31. P. Wyckoff, S. McLaughry, L. J. Tobin, and F. A. Daniel, TSpaces, *IBM Journal of Research and Development*, 37:3-Java Technology, pages 454-474, 1998. 37, 48

Context-Dependency in Internet-Agent Coordination

Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli

Dipartimento di Scienze dell'Ingegneria – Università di Modena e Reggio Emilia
Via Campi 213/b – 41100 Modena – Italy
{giacomo.cabri,letizia.leonardi,franco.zambonelli}@unimo.it

Abstract. The design and development of Internet applications can take advantage of a paradigm based on autonomous and mobile agents. However, mobility introduces peculiar coordination problems in multiagent-based Internet applications. First, it suggests the exploitation of an infrastructure based on a multiplicity of local interaction spaces. Second, it may require coordination activities to be adapted both to the characteristics of the execution environment where they occur and to the needs of the application to which the coordinating agents belong. In this context, this paper introduces the concept of context-dependent coordination based on programmable interaction spaces. On the one hand, interaction spaces associated to different execution environments may be independently programmed so as to lead to differentiated, environment-dependent, behaviors. On the other hand, agents can program the interaction spaces of the visited execution environments to obtain an application-dependent behavior of the interaction spaces themselves. Several examples show how a model of context-dependent coordination can be effectively exploited in Internet applications based on mobile agents. In addition, several systems are briefly presented that, to different extent, define a model of context-dependent coordination.

1 Introduction

The design and development of Internet applications can effectively take advantage of an agent-based approach. By conceiving and developing applications in terms of autonomous entities, capable both of reacting to changes in the environment and of proactively dealing with unexpected situations, one naturally deal with the intrinsic uncertainty and dynamicity of the Internet. In addition, by exploiting the notion of sociality typical of the agent-oriented paradigm, one can more easily decompose an application in terms of a multi-agent organization, where all the necessary interaction protocols find a first-order accommodation and definition.

Among the other characteristics, *agent mobility* [13, 23], i.e., the agents' capability of moving across the Internet while executing, is being recognized as a useful property for both the actual execution and the conceptual design of Internet applications. The

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 51-63, 2000.

physical movement of a software agent–i.e., the transfer of its code, data, and state–towards the nodes where the resources it needs to access are allocated can provide for saving network bandwidth, and for increasing both the reliability and the efficiency of the execution [15, 23]. The *logical* movement of a software agent, instead, refers to the fact that, in agent-based Internet applications, it is suitable to abstract the Internet as a multiplicity of *execution environments* [11] and to design applications in terms of agents that are aware of the distributed nature of the target and that *logically move* (when not even physically) in different execution environments during their execution.

The presence of mobility, together with the intrinsic openness of the Internet scenario, also introduces several problems in the design and development of Internet applications. Among the others, *coordination* problems, because of the need for agents to interact, communicate, and synchronize their activities, both with other agents (inter-agent coordination) and with resources on the hosting execution environments (agent-environment coordination). With regard to physical mobility, relying only on traditional coordination models (i.e., peer-to-peer and client-server) for global interactions between agents cannot always be feasible if agents can freely move at a world-wide scale [3]. With regard to logical mobility, the fact that an agent executes and interacts on different execution environments during its life has several implications on application design. On the one hand, one must take into account that each environment has its specific characteristics and security policies, which may somehow influence and/or constrain the behavior of an agent in coordinating with it. On the other hand, one may require the coordination activities of application agents to occur according to specific application needs, despite the different characteristics of the local environment and the different coordination rules there enacted.

To overcome the problems introduced by physical mobility of agents, it has already been argued that both inter-agent and agent-environment coordination can effectively rely on an infrastructure based on a multiplicity of independent interactions spaces (whether meeting points or tuple spaces), each associated to an execution environment and in charge of mediating both inter-agent and agent-environment interactions [3, 19]. This makes it possible for agents to coordinate, via the mediation of the local interaction space, without worrying about each other's position and name. In addition, by enhancing interaction spaces with the capability of dynamically programming their behaviors, one can make interaction spaces behave in different ways depending on which agent issued which interaction event. This characteristic can be used to support the logical mobility of agents, by making their coordination *context-dependent*. On the one hand, interaction spaces associated to different execution environments may exhibit different behaviors in response to the same interaction events. This enables to integrate in the form of new behaviors–transparently to agents–whatever needed environment-specific policy to rule the local coordination activities. On the other hand, on a given site, the same interaction events performed by agents belonging to different applications can lead to differentiated, application-dependent, behavior. Thus, agents can carry-on the code needed to implement and control an application-specific coordination policy and install it in the form of a new behavior into the interaction space of the visited site. In this way, agents can coordinate on the site being guaranteed that the interaction space will let them coordinate accordingly to their specific needs.

This paper is organized as follows. Section 2 briefly motivates the adoption of local interaction spaces in Internet applications based on mobile agents, by specifically focussing on tuple-based interaction spaces, and by introducing a simple case study application. Section 3 details the concept of context-dependent coordination and discusses, via several examples, how it can be effectively exploited. Section 4 briefly surveys several existing coordination systems and models that, to different extent, integrates a model of context-dependency. Section 5 concludes the paper, outlining open research directions.

2 From Global and Coupled to Local and Uncoupled Interactions

Physical mobility of agents introduces peculiar problems that discourage the generalized adoption of traditional coordination models, such as peer-to-peer message-passing and client-server, which enforce global and coupled interactions. First, the globality of the Internet scenario and, therefore, of the space of possible interactions, can make communications between distant agents inefficient and unreliable, also calling for complex infrastructure to deal with message forwarding [18], and lookup of agents' positions and names. Second, the intrinsic autonomy and dynamicity of agents clashes with the strict coupling (in terms of both synchronization of the activities and necessity of sharing a common name space) enforced by these models.

The alternative is to make agent coordination rely, whenever possible, on an infrastructure enforcing local and uncoupled interactions. On the one hand, interactions are to be confined within a local interaction space associated to the local execution environment. On the other hand, the local interaction space should act as the medium through which asynchronous and anonymous (i.e., uncoupled) interactions may occur. The mediation of the interaction space can make it possible for agents to indirectly interact with other agents disregarding their current positions and names, and for the local environment to make the interaction space act as the gate through which agent can access the local resources. Although different kinds of infrastructures can be conceived providing this characteristics and inspired by different coordination models (e.g., meetings [23] or event-channels [2]), we specifically suggest the adoption of an infrastructure based on independent tuple spaces [1, 4] associated to each execution environment. An agent on a site–via storing and retrieving of tuples in the local tuple space–can exchange data and knowledge messages with other agents, in an associative way, whoever and wherever these agents are, and whenever they will read (have written) the messages. In addition, an environment can publish in the form of tuples in the local space its public resources, to be accessed by locally executing agents.

Let us consider an application in which a mobile searcher agent roams an itinerary of Internet sites to access and analyze specific HTML pages. The searcher agent arrives at a site, retrieves the needed data, and continues its itinerary. However, if the searcher agent discovers the presence of other interesting sites (via HTML links) not

originally in its itinerary, it can clone itself and let the clones go to those sites. This can generate a tree of searcher agents, and makes it likely for two agents to arrive in the same site at different times. This situation must be avoided by coordinating the agents' movements: when on a site, a searcher agent must know if the site has already been visited or not by another searcher agent.

Since a searcher agent not only does not know where the other searcher agents are, but it also ignores who and how many they are, relying on peer-to-peer interaction between them is not feasible: searcher agents are forced to refer to a fixed, well-known, "home agent", acting as mediator for all searcher agents, introducing performance and reliability problems, and also complicating the application design. Instead, a tuple space infrastructure can be exploited in a very simple and elegant way, leading to a more efficient and simple application design. When a searcher agent arrives on a site, it simply checks the local tuple space for the presence of a tuple left by another searcher agent on a previous visit. If the tuple is not found, the site has never been visited and the incoming searcher agent is in charge of leaving a tuple in its turn, to mark its current visit.

In the above example, the tuple space–due to the associative access to tuples–can be effectively used also to access the information about the local HTML files. The administrator can write, in the tuple space, tuples in the form: (filename, extension, keyword, modification_time, size). Searcher agents, by their side, can retrieve information about local HTML files related to the argument "coordination", by asking for tuples matching: (string?, "html", "coordination", date?, int?).

3 Towards Context-Dependent Coordination

A model of local and uncoupled interactions not only fits the physical mobility of agents, but also invites thinking applications in terms of logical mobility. When an agent migrates to a new execution environment, its interaction space changes accordingly and is confined within the new environment. In other words, the agent's perceivable world changes as a consequence of its movement.

Even if the interaction model is the same independently of the environment, agents are likely to face several problems in proceeding with their coordination activities in all the different environments they visit. First, coordinating with the agents residing in a foreign environment may require facing all the typical problems of open systems, such as opportunistic behavior in interactions, heterogeneity of languages and protocols [24, 25]. Second, each environment has its specific characteristics and may somehow constrain the behavior of an agent in coordinating with it, for security or resource control reasons. Finally, despite the different characteristics of each environment, application agents may still require that their coordination occurs according to specific application needs.

In such a scenario, if the interaction space is simply an infrastructure for storing and forwarding of data and message, agents have to embed in their code all the intelligence needed to effectively handle their coordination activities in foreign execution environments. Alternatively, one can think about having the interaction

space itself in charge of the burden of handling all coordination activities, to meet either environment-specific needs, or application-specific ones, or both. However, this requires the interaction space to be somehow active and to enable a dynamic adaptation of its behavior in response to interaction events.

For instance, one can enhance the basic tuple-space model towards a *programmable tuple space model*: for any kind of access events, a new behavior in response to it can be programmed to override the basic behavior of the tuple space (i.e., storing and extraction of tuples based on a stateless and built-in pattern-matching mechanism). To this end, one must: *(i)* characterize the kind of access events of interest, in terms the identity of the agent performing it, the primitive used to access the tuple space, and the parameter tuple supplied in the primitive; *(ii)* express the new behavior (also called *reaction*) to be assumed by the tuple space in response to this kind of access events. Similar considerations may apply when considering the enhancement towards programmability of interaction spaces other than those based on the tuple space model [8].

The adoption of a programmable tuple space model in mobile-agent Internet applications, as well as of any model based on programmable and local interaction spaces, naturally leads to a model of *context-dependent coordination* (see Figure 1). On the one hand, the administrator of one site can adapt the behavior of the interaction space to enforce–transparently to agents–site-specific policies on the local interactions agents on a site, whatever the application they belong to. These laws can be different from environment to environment. On the other hand, the agents of a given application can spread their own application-specific coordination laws on the sites they visit, and these coordination laws will influence the coordination activities of all–and only–the agents belonging to the same application.

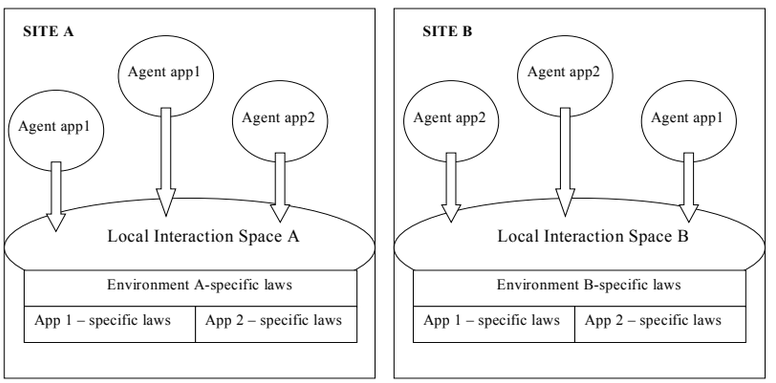


Figure 1. The Scenario of Context-Dependent Coordination

In the following of this section, we describe several cases for which programmable interaction spaces (specifically, programmable tuple spaces) can be effectively exploited in Internet-agent applications to lead to both environment-dependent and application-dependent coordination.

3.1 Environment-Dependent Coordination

Agents, while accessing a tuple space always with the same interface, can have the semantics of their coordination activities (as well as their perception of the environment) affected by the specific behavior programmed for the local tuple space. In other words, the same interactions can have different effects depending on the site in which they are performed. This can be used both to embed security policies in the tuple space, and to help agents entering the new environment without having to explicitly take in charge of all the issues implied in executing in and accessing a foreign environment.

Enforcing Security. When a site opens itself to the execution of mobile agents, it must be aware that malicious agents are likely to arrive at the site and undermine the integrity of its data and resources, which must be therefore protected from unauthorized accesses. Agents, on their side, may not be aware of the security policies adopted by an execution environment, so that their interactions with an environment are likely to issue a large number of security exceptions, to be handled by the agents themselves. However, if all the interactions with the environment are mediated by a programmable tuple space, the above problems can find a simple and elegant solution. In fact, the administrator can program the behavior of the tuple space so as to make any unauthorized access to the tuple space harmless without requiring the agent to handle exceptions. In the application example, if a searcher agent tries to extract a tuple representing a HTML file (while it has only the right to read that kind of tuples) a properly programmed tuple space can provide that tuple to agent without actually extracting it from the tuple space. As an additional example, a stateful reaction can be programmed to let searcher agents read only a specified amount of HTML tuples: once the specified amount has been extracted, the tuple space simply starts appearing to agents as if it does not contain further HTML tuples. In both examples, agents do not experience any exception when violating the access control policies of the environment: they simply perceive a different view of the tuple space content, while the local environment can maintain and enforce its security policies.

Handling Heterogeneity. Different execution environments can adopt different choices w.r.t. the format of their data and resources stored, as well as w.r.t. their representation in the local tuple space. Agents, in these cases, have to explicitly deal with heterogeneity and somehow discover how to fruitfully access the environment. When adopting a programmable tuple space model, a completely different perspective can be adopted. In particular, a tuple space can be programmed so as to react to the access performed by agents by somehow transforming the agents' request and making them homogeneous to the local representation. Therefore, on the one hand an agent can perceive a tuple space as if it were homogeneous to its expectations; on the other hand, the environment is not forced to change its tuple-based representation of the environment or duplicate it in different formats. In the application example, searcher agents look on a site for HTML files having the "html" extension, by asking for the corresponding HTML tuples. If the environment, by its side, stores HTML pages in files having the "htm" extensions, and shapes its tuples accordingly, a searcher agent,

unless intelligent enough, has no possibility of discovering the presence HTML pages of that site. Then, the administrator (which is supposed to know that most of the agents will look for "html" files) can modify the behavior of the tuple space so as to transform the agents' requests into requests for "htm" files, transparently to agents. In other words, the administrator modifies the matching mechanisms so as to make "html" match with "htm" in HTML tuples.

Supporting Open Interactions. Exploiting tuple space programmability to deal with heterogeneity issues naturally extends also to these cases in which a site is supposed to be open to host the execution of agents belonging to different applications and organizations, possibly heterogeneous in terms of supported languages and protocols, and nevertheless in need of coordinating with each other. Again, a tuple space can be used as a mediator, so as to support the coordination activities among a group of heterogeneous agents. Even more, a trusted site can be used to act as an impartial arbiter, in charge of controlling the interactions of agents with other, possibly self-interested, agents, that would be likely to be unfruitful or damaging the agents, otherwise. As an example still related to the information retrieval area, the tuple space can be used so as to control the behavior of opportunistic "advertising agents" that could have interest in diverting the research of searcher agents toward specific commercial sites.

3.2 Application-Dependent Coordination

Application designers can exploit the programmability of the tuple spaces in different ways. It can be used to facilitate the access to the information on the visited site, to support the exchange of complex knowledge between agents and to implement complex coordination protocols. More generally, application designers can exploit programmability of tuple spaces so as to adapt the interaction model to their specific, application-dependent, needs. Of course, since these modifications of the tuple spaces' behaviors are intended to meet specific application needs, some form of confinement must be provided by the tuple spaces to ensure that any application-specific behavior can affect only those access events performed by agents of that specific application.

Facilitating Access. Let us consider again the application example. To avoid duplicated work, one can think about having searcher agents put a "marker" tuple in the space of the visited sites. This makes later incoming searcher agents of the same application aware of the fact that the site has already been visited. Even better, the marker tuple can report the time of the visit: later incoming searcher agents, in this case, can simply detect which files have been modified since the previous visit and collect only the information that is updated w.r.t. that held by the earlier visiting agent. However, the basic pattern matching mechanism would force a searcher agent to retrieve all the tuples representing files and, subsequently, to select only those representing files which have been modified since the last visit. Therefore, the application calls for a different pattern-matching mechanism, to effectively support its inter-agent interaction needs, i.e., one that can make it possible to select all HTML

tuples in which the modification_time field is greater than the one specified in the template tuple. Of course, only those access events performed by agents of the application example must trigger the reaction leading to the new pattern-matching mechanism. That is, the specific pattern-matching mechanism has to influence only that application context.

Exchanging Knowledge. In the application example, putting a marker tuple in the space to be read by other application agents is a sort of knowledge exchange between agents of the same applications: "I have visited this site at time T and, if any agent ever reads that tuple, has to know that it has to analyze only those HTML files that has been modified since time T". However, when we designed the application, we already knew that the application was composed of multiple agents, which were likely to arrive on the same site at different times. Therefore, we designed each agent so as to explicitly look for marker tuples in the visited site. Let us now suppose, instead, that an agent does not know that there are other application agents that can possibly have visited the same site in the past. In this case, of course, an agent does not worry at all about acquiring this knowledge by checking for the presence of a marker tuple in the tuple space. Actually, this is a very common case in multi-agent applications: an agent acquires some sorts of new knowledge or expertise ("I know what information was in that site at time X") that can be very useful to the other agents to improve their work ("If I go to that site I have to analyze updated information only") and other agents, instead, cannot think at acquiring ("Why should I suppose that someone else has already visited a site?"). In the presence of a programmable tuple space model, the knowledge agents acquire can, in several cases, be embedded in the form of new tuple space behavior, that is meant to influence further accesses to the tuple space performed by other agents of the same application. In other words, an agent can install in a tuple space a reaction that reflects, in term of produced information, the knowledge acquired, to be transferred to other application agents. Via the reaction, other application agents can exploit this knowledge in a transparent way, without having to be explicitly informed and without being forced to ask for this knowledge. With regard to the above examples, an agent that has visited a site can install a reaction that returns, to other agents of the same application, only those files that have been modified since its last visit. In that way, another agent arriving there does not have to know or to worry about previous visits, because the reaction guarantees the avoidance of duplicated work. The above example shows how making coordination on a site application-dependent can lead to a very powerful model, which is likely to simplify inter-agent coordination and to enable dynamic exchange of knowledge without influencing at all the agent code.

Implementing Coordination Protocols. The burden of implementing a coordination protocol, whether involving agents interacting in a peer-to-peer way or indirectly via a not-programmable tuple space, increases the complexity of agents. In fact, agents are in charge of implementing in their code the capability of directly handling and controlling the proper execution of the coordination protocols. A better and cleaner solution would be to charge the interaction space itself (i.e., the tuple space) of the burden of controlling the coordination protocols. In particular, a programmable tuple

space can express the control part of the coordination protocols in terms of behaviors in response to access events. This can free agents from the duty of actually controlling the execution of the protocol, and provides for a clean separation of concerns between the "computational" task of application agents and the "coordination rules" required for the protocols of the application.

4 Systems and Models for Context-Dependent Coordination

In our opinion, the concept of context dependent coordination is a very general one. In this section, we firstly survey the approaches of some coordination systems for the Internet that–to different extents–integrate some sort of context-dependency. Then, we discuss how concepts strictly related to the context-dependency have been introduced in several proposals in the area of multi-agent systems engineering.

4.1 Coordination Systems

To our knowledge, none of the proposals in the area of coordination systems and architecture for agents and mobile agents explicitly introduces concepts somehow related to the one of context-dependent coordination. However, most of these systems implicitly define a context-dependent coordination model.

MARS (*M*obile *A*gent *R*eactive *S*paces), developed at the University of Modena and Reggio Emilia, and described in [4], implements a portable and programmable Linda-like coordination architecture for Java mobile agents, which naturally supports a context-dependent coordination model. Globally, the MARS architecture is made up of a multiplicity of independent programmable tuple spaces, each one associated to a node and accessed by (and only by) the agents locally executing in that node: in other words, in MARS, logical mobility of application agents has to necessarily imply their physical mobility. MARS tuple spaces can be independently programmed by associating reactions to access events, via *meta-level tuples* stored in a *meta-level* tuple space. A meta-level tuple represents a specific event (or class of access events) in terms of identity of the invoking agents, type of operation performed, type of tuple/template provided. When an access event performed on the base level MARS tuple spaces matches the meta-level tuple, the execution of a method of a specified Java object is triggered to replace the basic pattern-matching mechanism. Several meta-tuples can match a single access event, in which case all the corresponding reactions are composed and executed in pipeline. When a reaction method executes, it is provided with the information about the identity of the agents that has triggered the reaction, the operation it has performed, as well as the output of the matching mechanism (or of the previously executed reaction in the pipeline). The fact that a reaction can be associated to access events either when performed by agents with a specific identity or independently of the identity of the accessing agents enables for both environment-dependent coordination (the reaction applies to all agents), and application-dependent one (the reaction applies only to specific agents).

The *TuCSoN* model [19], developed in the context of a research project affiliated to MARS, adopts a very similar architectural model, and enhances it by making it possible for agents to refer to remote tuple spaces via URLs, as an Internet service. This enforces network-awareness and logical mobility without forcing physical agent mobility. With regard to the tuple space model, TuCSoN resembles MARS in its full programming capability of tuple spaces, although TuCSoN defines logic tuple spaces where both tuples and tuple space behaviors are expressed in terms of untyped first-order logic terms, and where unification is the basic pattern-matching mechanism.

The LIME systems attempts to face the problem of handling interactions in the presence of mobility in a more general way, also including mobile users and devices [22]. Each "agent", whether a software agent, an Internet node, or a physical mobile device, owns and carries a tuple space in its movement. The agent-owned tuple space is the only medium provided to the agent itself for interactions. Whenever the agent keeps in touch (i.e., gets connected) with another agent, the tuple spaces owned by each of the agents merge together, thus allowing to interact via the merged tuple space. A limited form of reactivity is provided to automatically move tuples across tuple spaces, and to notify agents about the events occurring in their tuple space. Although not explicitly mentioned by their designers, LIME naturally promotes a primitive form of context-dependent coordination: the effect of an agent accessing its own-tuple space can be very different depending on whether the tuple space is currently merged with other tuple spaces and which ones, that is, depending on which context the agent currently belongs to.

The model of Law-governed interactions, described in [16], addresses the problem of making peer-to-peer coordination within a group of non-mobile agents obey to a set of specified rules. An agent can initiate a group, and fix the rules to be respected by other agents willing to enter and interact within the group. These rules are typically security-oriented, and express which operations the agents are allowed to perform, and in which order. To enforce these rules, the model dynamically associates a controller process to each agent that joins the group. The controller process intercepts all messages to from the agent and, by coordinating with other controllers, checks their compatibility with the enforced rules. Although the Law-governed interaction model does not address in any way the problems related to agent mobility and does not explicitly identify environment-dependent and application-dependent coordination, it has the full potential for enforcing context-dependent coordination in complex systems of non-mobile agents interacting in a peer-to-peer way.

4.2 Models and Methodologies for Multi-agent Systems

The research areas of distributed artificial intelligence and of multi-agent systems are recently recognizing that interactions in agent systems–and thus the engineering of multi-agent systems–can no longer simply rely on the agent capability of communicating via agent communication languages and of acting accordingly to the need/expectations of each agent in the system. Instead, concepts such as "organizations" [7, 9], "organizational rules" [12, 25], "social laws" [17], "social order" [6], "active environments" [21] are receiving more and more attention, and are

leading to a variety of models and proposals. The idea underlying that variety of heterogeneous proposals is that, for the effective engineering of multi-agent systems, higher-level, inter-agent, concepts and abstractions must be defined to explicitly model the environment in which agent live and have to interact, and the associated laws.

As an example, several researches show that engineering a complex multi-agent organization does not simply require the identification of the roles to be played by agents, but it also requires the identification of the global constraints under which the interactions between roles in the organization have to occur. The identified global constraints can then be hardwired into the agents to limit their possible actions [17], or they can be used to select the most suitable organizational structure for the system [25].

As another notable example, complex (and useful) global behaviors of the multi-agent system can emerge from the interactions of very simple agents with the environment [20, 21]. Agents put and sense pheromones in the environment, and act accordingly to the concentration of pheromones in given section of the environment. The environment, by its side, makes pheromones vanish with time, according to specific laws. The global behavior that emerges depends strictly on the laws upon which pheromones vanish: in other words, the environment plays an active role by influencing the overall behavior of the system via an imposition of environment-specific rules.

The concept of context-dependent coordination introduced by this paper is likely to facilitate the definition and the implementation of complex multi-agent systems on the Internet. On the one hand, concepts such as organizational rules can be easily enforced by programming of the interaction spaces, without having to hardwire them into agents or into the organizational structure. On the other hand, a programmable interaction space represents the natural implementation on the Internet of an active environment upon which to rely for controlling the emergence of global behaviors.

5 Conclusions and Work in Progress

In this paper, we have introduced the concept of context-dependent coordination for mobile agents, and have shown how this concept can be effectively exploited in Internet applications. Defining a framework in which agent coordination in the Internet may easily be represented and controlled, and may be tuned to the specific needs of both applications and environments, can simplify application design and management, and may represent an important driving force towards the successful deployment of Internet applications based on mobile agents.

Several issues not addressed by this paper still needs to be analyzed for the proposed concepts to be effectively usable and used. In particular: *(i)* coordination infrastructures must be somehow integrated with Agent Communication Languages [10], to let agents interact in terms of high-level, knowledge-based, information; *(ii)* roles and role models [14], due to their importance in the analysis and design of multi-agent applications, have to be somehow integrated in the model; *(iii)* specific

software-engineering methodologies [24, 25] must be defined to help in developing agent-based applications and in clearly devising whether and how to exploit agent mobility and context-dependency.

All the above issues are the current interest of our research group. In addition, we aim at defining a more general and formal notion of context-dependent coordination, possibly exploiting already defined formal models for mobile systems [5].

Acknowledgements

This work was supported by the Italian National Research Council (CNR) in the framework of the project "Global Applications in the Internet Area: models and programming environments".

References

[1] S. Ahuja, N. Carriero, D. Gelernter, "Linda and Friends", IEEE Computer, Vol. 19, No. 8, pp. 26-34, August 1986.

[2] J. Baumann, F. Hohl, K. Rothermel, M. Straßer, "Mole - Concepts of a Mobile Agent System", The World Wide Web Journal, Vol. 1, No. 3, pp. 123-137, 1998.

[3] G. Cabri, L. Leonardi, F. Zambonelli, "Mobile-Agent Coordination Models for Internet Applications ", IEEE Computer, Vol. 33, No. 2, pp. 82-89, February 2000.

[4] G. Cabri, L. Leonardi, F. Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents", IEEE Internet Computing, Vol. 4, No. 4, pp. 26-35, July-August 2000.

[5] L. Cardelli, D. Gordon, "Mobile Ambients", Foundations of Software Science and Computational Structures, LNCS No. 1378, pp. 140-155, 1998.

[6] C. Castelfranchi, "Enginnering Social Order", 2000, in this volume.

[7] Y. Demazeau, A.C. Rocha Costa, "Populations and Organizations in Open Multi-Agent Systems", 1st National Symposium on Parallel and Distributed Artificial Intelligence", 1996.

[8] E. Denti, A. Natali, A. Omicini, "On the Expressive Power of a Language for Programmable Coordination Media", Proceedings of the ACM Symposium on Applied Computing, ACM, 1998.

[9] J. Ferber, O. Gutknecht, "A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems", 3rd International Conference on Multi-Agent Systems, Paris (F), IEEE CS Press, pp. 128-135, July 1998.

[10] T. Finin et al., "KQML as an Agent Communication Language", 3rd International Conference on Information Knowledge and Management", November 1994.

[11] A. Fuggetta, G. Picco, G. Vigna, "Understanding Code Mobility", IEEE Transactions on Software Engineering, Vol. 24, No. 5, pp. 352-361, May 1998.

[12] N. R. Jennings, "On Agent-Based Software Engineering", Artificial Intelligence, Vol. 117, No. 2, pp. 277-296, 2000.

[13] N. M. Karnik, A. R. Tripathi, "Design Issues in Mobile-Agent Programming Systems", IEEE Concurrency, Vol. 6, No. 3, pp. 52-61, July-September 1998.

[14] E. Kendall, "Role Modelling for Agent Systems Analysis, Design and Implementation", 1st International Symposium on Agent Systems and Applications", Palm Springs (CA), IEEE CS Press, October 1999.
[15] D. B. Lange, M. Oshima, "Programming and Deploying Java™ Mobile Agents with Aglets™", Addison-Wesley, Reading (MA), August 1998.
[16] N.H. Minky, V. Ungureanu, "Law-Governed Interaction: A Coordination & Control Mechanism for Heterogeneous Distributed Systems", Draft Technical Report, Department of Computer Science, Rutgers University, 2000, available at http://www.cs.rutgers.edu/~minsky/pubs.html. to appear in ACM Transactions on Software Engineering and Methodologies.
[17] Y. Moses, M. Tenneholtz, "Artificial Social Systems", Computers and Artificial Intelligence, Vol. 14, No. 3, pp. 533-562, 1995.
[18] A.L. Murphy, G.P. Picco, "Reliable Communications for Highly-Mobile Agents", 1st International Symposium on Agent Systems and Applications", Palm Springs (CA), IEEE CS Press, October 1999.
[19] A. Omicini, F. Zambonelli, "Coordination for Internet Application Development", Journal of Autonomous Agents and Multi-Agent Systems, Vol. 2, No. 3, pp. 251-269, September 1999.
[20] H. V. D. Parunak, "Go to the Ant: Engineering Principles from Natural Agent Systems", Annals of Operations Research, Vol. 75, pp. 69-101, 1997.
[21] H. V. D. Parunak, S. Brueckner, J. Sauter, R. S. Matthews, "Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies", 2000, in this volume.
[22] G.P. Picco, A.M. Murphy, G.-C. Roman, "LIME: Linda Meets Mobility, 1999 International Conference on Software Engineering, Los Angeles (CA), ACM Press, 1999.
[23] J. White, "Mobile Agents", in J. Bradshaw ed.: Software Agents, AAAI Press, Menlo Park (CA), pp. 437-472, 1997.
[24] F. Zambonelli, N. R. Jennings, A. Omicini, M. J. Wooldridge, "Agent-Oriented Software Engineering for Internet Applications", in Coordination of Internet Agents: Models, Technologies and Applications, Springer, 2000, to appear.
[25] F. Zambonelli, N. R. Jennings, M. J. Wooldridge, "Organizational Abstractions for the Analysis and Design of Multi-agent Systems", 1st International Workshop on Agent-Oriented Software Engineering, LNCS, 2000, to appear.

Coordination Issues in Multi-agent Event Data Processing

Christoph Koch[1] and Paolo Petta[2⋆]

[1] European Organization for Nuclear Research (CERN),
CH-1211 Geneva, Switzerland
christoph.koch@cern.ch
[2] Austrian Research Institute for Artificial Intelligence,
Schottengasse 3, A-1010 Vienna, Austria
paolo@ai.univie.ac.at

Abstract. We present a multi-agent systems approach to distributed event-data processing as it is pervasive in scientific computing environments. The task investigated is the one of configuration and execution of event-data processing pipelines to be assembled from single computational services (agents) that perform an asynchronous mapping of streams of inputs to streams of outputs, where the specification is given by characterizing the final output of a pipeline.
Comprehensive declarative descriptions of the capabilities of single agents in such systems can be shown to be computationally intractable because of the complexity of the mapping between inputs and outputs of individual agents. We therefore investigate the consequences of circumventing this problem by publishing just the output capabilities of agents, performing the transformation of output requirements to input requirements opaquely within individual agents, and utilizing recursive runtime contracting to configure complete data processing pipelines.
The information loss entailed by this kind of information propagation opens up the possibility of pipeline misconfigurations that in turn lead to runtime exceptions when constraints between interfaces that were not explicitly enforced by the published capability descriptions are violated. We characterize the ensuing coordination needs and related design requirements for such kinds of multi-agent systems and propose the introduction of social laws as a promising principled solution approach to be further researched.

1 Introduction

Distribution in scientific computing is more than a way to satisfy the ever-increasing computing power and scalability requirements. It is a software engineering paradigm required in today's huge and globally distributed scientific undertakings. As collaborations of scientific institutes without centralized control abound, monolithic software development which relies on agreements on

⋆ The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 64–77, 2000.

tools, frameworks, libraries, and shared views on software design is losing ground against approaches which allow further decoupling of programs without loss of interoperability. Furthermore, scientific computing setups operate in experimental environments, which means that it has to be possible to add and exchange software components and to experiment with software configurations while systems are running.

For these reasons, the full decoupling of the interfaces of software programs, and their description and dynamic matching is desirable in large scientific computing settings. While intelligent problem solving capabilities or highly complex patterns of collaboration are not among the main requirements of this domain, software components nevertheless have to act flexibly and socially. All this points at the appropriateness of a multi-agent systems approach. While present research is being carried out in the framework of examining opportunities for the application of multi-agent technology to the processing and analysis of high-energy physics event-data at CERN, it may be noted that similar requirements are coming about rapidly in the area of ubiquitous wireless networking, e.g. in the private domestic area, and otherwise.

The Problem Scenario: Configuration of Event Data Processing Pipelines

This paper deals with event-data processing, which is pervasive in scientific environments. This kind processing is inherently asynchronous, as data originate from a source — an experimental setup — which is outside of direct human control and which produces event-data at unknown time points and in unknown quantities. These data have to be processed as they arrive, and the asynchronicity cannot be circumvented by e.g. storing the event-data in a database where analysing programs could retrieve them at their own pace, because partial processing results usually have to be fed back into the control of the experiment. This only leaves room for local storage of data cover limited periods of time by the single processing units.

For any given processing element along the event-data stream pipeline, input of whatever size does not necessarily imply generation of any data output. Therefore, asynchronicity has to be considered not only with respect to flow of control but also with respect to flow of data. The processing tasks under consideration thus differ substantially from the model generally used in planning and coordination, as tasks cannot be considered to accept a “unit of input” (however large) and produce a “unit of output”. Furthermore, in some processing tasks the generation of output is triggered by an external control signal that is independent of the incoming event-data stream. Because of that, we regard this domain substantially different from work in supply chain management (e.g. [18]) or agent-based business process management (such as [13]). Furthermore, due to the need for clean decoupling and flexibility of the infrastructure, more home-grown approaches of the scientific computing community, such as high-performance computational steering (see [21] and [16]) are also not applicable, nor are distributed scientific problem solvers such as NetSolve [6].

In order to facilitate the matching and subsequent collaboration of software agents that communicate via streams of asynchronously occurring events, capability descriptions are required. Such descriptions often end up requiring very high expressive power, so that reasoning over them becomes undecidable under the known formalisms. One particular source of complexity in our case is the need to describe ordering relationships across many events of an input stream of an agent, which are required for the agent to be able to stay synchronized with its input streams. Relatedly, ordering relationships are also needed for the output streams of such asynchronously operating agents, so that the capability descriptions can be matched with those of other agents that will use the output further down the line.

This paper is structured as follows. Section 2 discusses distributed event-data processing in more detail. In particular, the intrinsic asynchronicity of event-data processing algorithms is explained and the time-related concepts that structure it are outlined. In Section 3 we discuss the problems with the application of capability description encountered in this domain which are due to the particular task structure, namely the requirement to model streams of events, relationships between events, and invariants and other relationships between streams. As we will explain, known description formalisms do not allow to perform the necessary modelling tasks due to limitations in terms of complexity or undecidability. This essentially forces us to move part of the very matching process between components from the time of the configuration of the data processing pipeline to the time of actual data processing: section 4 discusses the handling of exceptional situations when whilst processing agents discover that their publishers or subscribers are mismatched to them. This entails some requirements with regard to the tracking of wrongly published information that requires collaboration among the agents in the system. We discuss a simple proposal for this coordination task based on the contract net protocol [19], and motivate the appropriateness of utilizing such a *mutual* selection mechanism.

Finally, in Section 5, we give some conclusions and formulate our next research goal of investigating the suitability of applying social design principles to enable collaboration in this novel multi-agent systems application domain.

2 Distributed Event-Data Processing Systems

As shown in Figure 1, a system of event-data processing agents works much like a pipeline. Agents receive data, process them, and pass on the results. From left to right, the amount of data propagated between nodes decreases, as the data are refined and filtered for analysis by humans, for storage in databases for later use, or to be fed back into the control of the scientific experiment. In terms of Figure 1, such a pipeline is usually configured from the right to the left (see top of Figure 1), as a user or agent determines a processing need and asks the system to *provide* the necessary data. In the following, we point out two important characteristics of such systems: the asynchronicity of inputs and outputs of any

processing unit, and the particular relevance of ordering relationships within the event data stream as prerequisites for the processing itself.

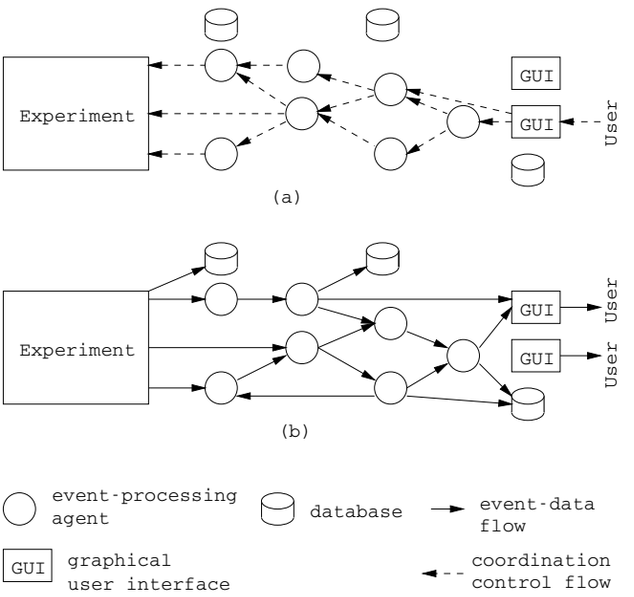


Fig. 1. Configuration (top) and execution (bottom) of a event-data processing pipeline

2.1 Asynchronicity of Inputs and Outputs

In this paper we use a simplified model of event-data processing algorithms which are executed by tasks inside the agents of our system (see Figure 2). We assume that an event-data processing algorithm gathers incoming data (possibly doing some preprocessing) until some internal condition is triggered, whereupon it does some concluding computation (a simple example would be computing the average of the collected values) and passes on the results to its subscribers. The internal condition is usually the end of a specific interval in the event-data stream over which a certain analysis computation has to be performed. The relevant time intervals are specific to the experiments. These may be concepts such as burst, spill, run, data taking period, or day. The output of such an algorithm thus is generated asynchronously with respect to the incoming data stream, and there is not necessarily a one-to-one relationship between chunks of input and results. This property has to be appropriately reflected in any description of the capabilities of such an algorithm.

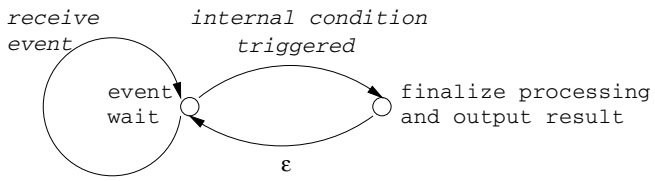


Fig. 2. An event-data processing algorithm

2.2 Required Data Stream Descriptions

For an agent to be able to detect the internal conditions triggering the production of output, it must assume that the stream meets certain constraints, such as events to be ordered by their timestamp, or more weakly, that all events of a given burst are received in succession without interference of data from another burst.

Consider the following pseudocode example for an algorithm that produces summary information over the events of a burst – most analysis algorithms work according to such a schema:

```
int previous_burst_index = -1; // the first burst has index 0
list<Event> events_of_this_burst;
while(true) { // forever
 read incoming event e; // blocks until event arrives
 assert(e.burst_index >= previous_burst_index);
 if(e.burst_index > previous_burst_index) { // new burst
    result_type result=do_the_computation(events_of_this_burst);
    send result to subscribers;
    events_of_this_burst.pop_all();
    previous_burst_index = e.burst_index;
 }
 else { // still the same burst
    events_of_this_burst.push_back(e);
 }
}
```

This algorithm implicitly assumes that the incoming events are ordered at least by burst (i.e., data of single events making up a burst need not be received in any particular sequence). This illustrates the desirability of an explicit description of such ordering requirements, so that they can be considered in the matching of components during configuration of whole event-data pipelines.

For example, an ordering by burst entails an ordering by run or data taking period: such constraints can determine the qualification of an agent for a given place in the assembly of a pipeline. More generally speaking, the description of orderings in such applications defines an additional orthogonal dimension of dependencies between software programs in such computing domains,

much as structural coupling (communication languages), shared conceptualizations (knowledge representation), and communication protocols and conversation policies. The constraints over streams seem to be an integral characteristic of event-based computing, and are therefore of broad relevance. To make such distributed systems open, the constraints over event streams have to be made explicit. We therefore next turn our attention towards the identification of techniques allowing the description of such capabilities and their processing.

3 Evaluation of Approaches to Interface Description

In this section we survey some representative approaches to description of interfaces of modules of distributed systems, ranging from mere syntactic matching to semantic capability descriptions, against the background of the task at hand, configuration of event-data processing pipelines. We motivate our resulting decision to reduce the amount of information published about single interfaces and defer part of the matching tasks to runtime, thereby ensuring computability at the cost of limiting the capacity to reason over the interface description employed.

In distributed object computing, interoperability with good decoupling can be achieved through technologies such as CORBA in combination with interface description languages such as IDL. But such languages have a serious drawback: they only allow capabilities to be matched by signature, but not by their semantics [3]. At the other extreme, languages such as EIFFEL [15], with its "Design by contract" approach which allows to specify pre- and postconditions as well as invariants, or Z have the drawback that their formalisms are so generic that reasoning with them is undecidable in general. In the case of EIFFEL, this entails that conditions are checked at runtime (when conditions are violated, exceptions will be thrown) but cannot be used for the reasoning over descriptions of software.

However, the reasoning over capability descriptions is necessary for matching software components, and more than the signature matching of IDL is required when the system is highly heterogeneous (such as when a large number of mostly independent developers write the software components), or when sophisticated reasoning such as configuration planning has to be performed in oder to schedule event processing pipelines, perform load balancing, or determine input events to be replayed from databases across a system of agents to compute results according to a given specification.

Let us first have a look at a capability description in first-order logic, which allows to fully describe the desired characteristics of the following example. (This example uses CDL [22] as "outer" language, with first-order logic in the notation of KIF as the "inner" language.)

```
(capability
 :action computeSomeSummaryOfStreamInterval
 :input (and (Stream ?is)
             (forall ?e (-> (member ?e ?is) (BeamEvent ?e))))
```

```
:input-constraints
 ;; input events have to be ordered by burst
 (forall (?e1 ?e2) (-> (and (member ?e1 ?is) (member ?e2 ?is)
                            (< (index ?e1) (index ?e2)))
                       (<= (burst ?e1) (burst ?e2))))
 :output (and (Stream ?os)
  (forall ?o (-> (member ?o ?os) (SomeSummaryConcept ?o))))
 :output-constraints
  (and
   ;; output events will be ordered by burst
   (forall (?o1 ?o2) (-> (and (member ?o1 ?os) (member ?o2 ?os)
                              (< (index ?o1) (index ?o2)))
                         (< (burst ?o1) (burst ?o2))))
   ;; there will be an output event for each burst for which
   ;; input events were supplied
   (forall (?b ?e) (-> (and (member ?e ?is) (burst ?b ?e))
                       (exists ?o (and (member ?o ?os)
                                       (burst ?b ?o))))) ))
```

This example describes a capability of an agent to collect event-data and, upon certain events (every time a burst is over), to perform a computation over the collected data and send off a result to subscribed agents. For such a program to be able to make sense of its input, there have to be certain agreements about the relationships between events arriving in a stream. For example, the above capability description says that the program requires the incoming events to be ordered by their event index (in ascending order), so that it will know that if the burst index increases from one event to the next, a burst has ended, which may trigger a special computation.

In the matching process between two agents, these constraints have to be satisfied for the agents' interfaces to be compatible. The requirement to describe such an ordering is as much part of the interface of a component as the agent communication language, information model and protocols it understands. Relying on a correct ordering without making this explicit is an impediment to interoperability, as it may lead to communication problems between agents that are hard to debug.

As shown above, full first order logic allows to conveniently express this kind of capability description. Unfortunately, subsumption in first-order logic is undecidable in general.

In the field of Description Logics (e.g. [10] or [11]), researchers have attempted to constrain logics such as first-order logic to derive structured logical languages which are decidable (or even allow efficient reasoning) while at the same time being sufficiently expressive for various applications. As discussed in Section 2, distributed event-data processing requires to describe constraints to be taken into account across streams of events. We will outline that classic description logics in the style of KL-ONE [5] or the $\mathcal{ALC}$ family of languages (e.g. [10] or [17]) are unsuitable for description problems of this kind. As we will discuss, the

known description logics formalisms, when extended by the constructs necessary to express these ordering constraints, become undecidable.

In a nutshell, description logics, when used for capability description [3], are meant to provide a convenient way for defining inputs and outputs of agents, together with invariants that can be useful for AI planning. For example,

```
(and SomeMethod (the in Amplitude) (the out Amplitude)
     (same-as in.burst out.burst))
```

would declare a capability *SomeMethod* that accepts an object of type *Amplitude* and produces an object of the same type, where the *burst* attributes of the input and the output object are assured to have the same value (depending on the agreements in the system, this will most probably only be the case after the method was executed). The above example uses the syntax of CLASSIC [2].

The main problem with this approach is that we cannot model constraints over streams of events in the sense we need it unless we extend our description logic by the possibility to use coreference constraints (same-as) over not necessarily single-valued roles (features) and to reason over integers. Unfortunately, both of these requirements independently entail undecidability of subsumption (e.g. [4]), even in the otherwise simplest description logics.

While it is possible to do without these extensions in certain cases, it is not possible to circumvent this problem in general. For example, by modelling streams as lists (where events are chained via successor features), single-valued same-as suffices, as in the following example:

```
:in (and Stream
   (all member ElectronEvent)
   (all member (same-as successor.evtindex evtindex.successor)))
```

which enforces that consecutive event indexes are increasing by one. Apart from the fact that this can be seen as an inelegant solution, such an approach does not work in general, and it does not allow us to define constraints between streams.

Description Logics have been extended to support transitive roles (e.g. [9] or [12]), which allows to describe streams. Reasoning with these logics has high computational complexity (at least PSPACE-completeness), and it does not allow to deal with integers or orderings. Temporal description logics like $\mathcal{TL} - \mathcal{ALCF}$ [1] allow to describe relationships such as orderings between events, but one cannot express that events belong to streams, and so one cannot build relationships between streams (as it is necessary for invariants). Furthermore, these logics again are subject to very high reasoning complexity (e.g, subsumption in $\mathcal{TL} - \mathcal{ALCF}$ is EXPTIME-complete [1]).

Unfortunately, there seem to be only three alternatives to full description:

- One could extend the outer capability description language by primitives for expressing orderings and make implicit that all inputs and outputs are streams, so that descriptions of inputs and outputs are only made in terms of

events. This entails some coupling of the interfaces of the processing agents with the capability description mechanism and requires all agents to be event-driven. This is impractical, as a practical scientific computing environment e.g. also has to rely on an information integration infrastructure, in which agents are not dealing with streams of event-data.
– One could just send special events at certain times for synchronization, allowing to drop the task of describing constraints over streams as discussed, but in order to cater for openness and decoupling, the rules for when such events are triggered would have to be described as well, leading to an equally hard problem.
– As the final alternative, a more shallow method for capability description can be adopted and the remaining matching tasks deferred to runtime. Unfortunately, since the constraints on the relationships and orderings of events would have to be expressed in a description language at runtime as well (which would entail the problems we want to avoid), we could not always achieve a perfect match. Instead, agents would have to keep track of their input and trigger exceptions when they observed violations of their unpublished constraints. The possibility of such circumstances requires inclusion of repair policies in the communication protocols employed so as to retry affected processing tasks.

In the remainder of this paper, we will further investigate this third method and consider enforcing the full constraints of interfaces between event-data processing algorithms only at runtime. As the interfaces of agents are not published to the necessary extent, we will not be able to reason over these descriptions.

4 Coordination

In an event-data processing society, agents organize around publish-subscribe data
streaming relationships. Therefore, it is mostly the creation and termination policies of these relationships that require interesting coordination efforts. In addition, the exception handling that was motivated in the previous section also requires coordination. We use the contract net protocol [19] for the initiation of publish-subscribe relationships. This kind of *mutual* selection is necessary, as it is not possible to tell just from the reduced shallow capability descriptions whether an agent is actually well-suited for a particular streaming task. Also, there is some variable cost associated with a collaboration relationship (which we associate based on load balancing considerations), and different agents may be capable of providing the same data stream at different costs. In this section, we will try to address the problems of the following two use cases:

– An agent wants to subscribe to a data stream starting from a past time point. Here, an agent will have to rely on those agents that can provide exactly those kinds of event-data that the agent wants to subscribe to to propagate the requests to their own suppliers, who will again propagate it

to their suppliers, and so on (in other words, there will be a cascade of contracts). This has to be done because the shallow capability descriptions do not cover how inputs and outputs of agents correspond to each other, and which event-data have to be replayed into the system and from where. This problem is addressed in subsection 4.1.
– An agent mismatch is detected at data processing time. An exception handling behavior has to be executed to reinitiate coordination to find a new supplier of event-data. We discuss this issue in subsection 4.2.

4.1 Contracting in a Society of Event-Processing Agents

Due to our inability to express explicitly the capabilities of agents in appropriate descriptions, the relationships and invariants between their inputs and outputs stay implicit in the agents, and these have to provide additional help when an agent tries to contract for a certain stream of event-data: While we cannot express the relationships of inputs to outputs of individual agents in descriptions, the agents themselves are aware of these relationships and thus able to create contracts for the incoming requests. As already mentioned, the propagation of contracts may mean that a request is recursively pushed through a whole pipeline of event-data processing agents (see Figure 1). As connections between agents are always initiated from agents "downstream" (closer to the end-user) looking for agents further "upstream", capability descriptions solely need to cover agent outputs, i.e., to state what kinds of events an agent may produce—with the omission of ordering constraints, which, as discussed, cannot be expressed.

To subscribe to a certain kind of stream of event-data, an agent starts the standard contract net protocol to ask those agents who in principle could produce such a stream for bids. Bidding agents that are not too busy reply with a cost, which helps to avoid inefficiencies such as choosing an inactive agent that would have to connect to providers over another agent that is already producing this kind of events and is not overloaded. Determining whether an agent is able to send a bid may require it to start contracting for suppliers itself. For example, in the case where a stream has to start with a past time point, suppliers for old event-data have to be found first.

4.2 Exception Handling

As discussed, we run into difficulties when trying to identify a formalism for capability description which allows to express ordering constraints like those discussed in Section 2. An agreement by reasoning over such constraints can only be achieved itself if the content ("inner") part of the communication language can express such requirements—and the problem stays as hard as it was for matching by capability description. So, our problem is *not* solved by making capability matching more shallow and moving part of the matching to runtime *alone*. Rather, we furthermore explicitly allow for the *failure* of an agent while the data producing computation is already ongoing. In case of the pseudocode example of Section 2, the program would generate a runtime exception when it

finds that the originator of the input stream just sent an event that was older than the previous one (the code contains an assertion that would fail under these circumstances). This is not a very sophisticated or desirable solution by itself, but the only way to avoid the need to describe and reason over these orderings (and such run-time exceptions are in fact what for example EIFFEL does when its programming contracts are violated).

Such a run-time approach has the following consequences on the multi-agent system's architecture and coordination requirements, in case of an agent of a set-up pipeline throwing an exception:

- The current chain of providing agents upstream of the failing agent has to be undone (if the agents were connected by some kind of contracting, there has to be a way to backtrack to the contracting phase, releasing the current commitments), and the failing agent (that threw the exception) has to tell its subscribers that the information it sent possibly is invalid, as wrong orderings of events may have made the agent believe that a certain meaningful interval in the event stream was completed and led it to the emission of incomplete or premature results.
- One or more different supplying agent(s) for the failing agent that match the incomplete shallow capability description and were not tried yet have to be identified. As explained in section 4.1, this may lead to an "upstream" cascade of contracting.
- The failing agent has to reinitiate the subscription of its desired event stream starting from a past time point, as time (as well as events) has passed while the agent was connected to the wrong supplier.
- Since exceptions only occur when there is an actual mismatch of processing algorithms (whose immutable requirements have been violated), the connecting of specific (types [23] of) agents for a certain kind of streaming relationship should never be repeated once an exception has occurred.

5 Conclusions and Future Research

In this paper, we have introduced event-data processing as an area of interest to the MAS community. The peculiar structure of tasks in this domain and the identified limitations in applicability of established techniques have led us to propose a solution that distributes responsibility for ensuring the computation of correct results to all agents involved in a given event processing pipeline. Among the interesting novel questions that arise as a consequence, a prominent one is how to organise a principled approach to designing this kind of multi-agent systems. Alternatives include:

- a dominant coordination method, such as blackboard and GPGP [8], which mostly predetermine the overall design approach to be taken;
- an approach with an emphasis on organizational structuring: Clearly, in a mostly pipelined architecture, there are many pairs of agents that never

communicate with one another at all. One could for example build an organizational hierarchy from down- to upstream, with the agents "nearer" to the human user being superordinate to those closer to the experiment readout, partitioning sets of agents so as to prevent mismatches. Unfortunately, this would hamper system openness, and make matching of agents a case-by-case design task for human developers;

– social laws, and a holistic description of the system [7]: For example, one could formulate many important aspects of the discussed system by means of three high-level social rules:
  - *The system has to keep the event-data processing alive*: principal operation, coordination;
  - *the system has to be parsimonious with resources*: load balancing and responsible management of resources. For example, this will govern the use of cost in contracting to achieve a reasonable behaviour with regard to handling load, but it will also allow for agents that are overloaded to receive help from other agents);
  - *the system has to cooperatively track and resolve wrong information*: this came into the picture when we discussed exception handling, but in general this is an important aspect of such software systems in which the feasibility of algorithms can often only be found out empirically—frequently, only the results obtained when deploying an analysis method in actual experiments can show whether an algorithm produced the aspired results or whether its results have to be rejected. This social law entails that, when an agent is informed that some information it received was wrong, it will take local action but will also inform all other agents which it propagated related information to. Furthermore, if an agent finds that some information it produced was wrong, it will not try to keep this secret.

It is this latter direction that we intend to further discuss at the workshop and to use as a guideline for further research, relating our particular scenario to other research activities in the area of open multi-agent systems, such as [14].

References

1. Artale, A., Franconi, E.: A Temporal Description Logic for Reasoning about Actions and Plans. *Journal of Artificial Intelligence Research (JAIR)* **9**:463-506, December 1998. 71
2. Borgida, A., Brachman, R. J., McGuinness, D. L., Resnick, L. A.: "CLASSIC: A Structural Data Model for Objects". *Proc. of the 1989 ACM SIGMOD Int'l. Conf. on Management of Data*, pp. 59–67, June 1989. 71
3. Borgida, A., Devanbu, P.: "Adding more 'DL' to IDL: towards more knowledgeable component inter-operability". *Proc. of ICSE'99* (1999). 69, 71

4. Borgida, A. and Patel-Schneider, P. F.: "A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic". *Journal of Artificial Intelligence Research (JAIR)* **1**:277–308 (1994). 71
5. Brachman, R. J., Schmolze, J. G.: An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science* **9**:171–216 (1985). 70
6. Casanova, H., Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems Server for Solving Computational Science Problems. Tech. Report CS-95-313, Univ. of Tennessee, Nov. (1995). 65
7. Ciancarini, P., Omicini, A., Zambonelli, F.: "Multiagent System Engineering: The Coordination Viewpoint". In Jennings N. R., Lesperance T. (Eds.): *Intelligent Agents VI — Proc. of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, LNAI 1757, Springer-Verlag (2000). 75
8. Decker, K., Lesser, V.: "Generalizing the Partial Global Planning Algorithm". *Journal on Intelligent Cooperative Information Systems* **1**(2):319–346 (1992). 74
9. De Giacomo, G., Lenzerini, M.: "A Uniform Framework for Concept Definitions in Description Logics". *Journal of Artificial Intelligence Research (JAIR)* **6**:87–110 (1997). 71
10. Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.: "Reasoning in Description Logics". In Brewka G. (ed.) *Principles of Knowledge Representation and Reasoning; Studies in Logic, Language and Information*, CLSI Publications, pp. 193–238 (1996). 70
11. Franconi, E.: Description Logics Course Web Page. Available at `http://www.cs.man.ac.uk/~franconi/dl/course/` 70
12. Haarslev, V., Möller, R.: "Expressive ABox Reasoning with Number Restrictions, Role Hierarchies, and Transitively Closed Roles". In Giunchiglia, F. and Selman, B. (Eds.) *Proc. of 7th Int'l. Conf. on Principles of Knowledge Representation and Reasoning (KR'2000)*, Breckenridge, Colorado, USA, April 2000. 71
13. Jennings, N. R., Norman, T. J., and Faratin, P.: "ADEPT: An Agent-based Approach to Business Process Management". *ACM SIGMOD Record* **27**(4):32–39 (1998). 65
14. Klein, M., Dellarocas, C.: "Domain-Independent Exception Handling Services That Increase Robustness in Open Multi-Agent Systems". ASES Working Report ASES-WP-2000-02. Cambridge MA USA, Massachusetts Institute of Technology (2000). 75
15. Meyer, B.: *Object-oriented Software Construction*. Prentice-Hall (1989). 69
16. Parker, S., Weinstein, D., and Johnson, C.: "The SCIRun Computational Steering Software System". In Arge, E., Bruaset, A., and Langtangen, H. (Eds.) *Modern Software Tools in Scientific Computing*, pp. 1–44. Boston: Birkhauser Press, (1997). 65
17. Patel-Schneider, P. F., Swartout, W.: "Description Logic Knowledge Representation System Specification". KRSS Group, ARPA Knowledge Sharing Effort, (1993). 70
18. Sadeh, N., Hildum, D., Kjenstad, D., and Tseng, A.: "MASCOT: An Agent-Based Architecture for Coordinated Mixed-Initiative Supply Chain Planning and Scheduling". In *Proc. 3rd Int'l. Conf. on Autonomous Agents (Agents'99) Workshop on Agent-Based Decision Support for Managing the Internet-Enabled Supply Chain*, Seattle, (1999). 65
19. Smith, R. G.: "The contract net protocol. High-level communication and control in a distributed problem solver". *IEEE Transactions on Computers*, C–29(12):1104–1113 (1980). 66, 72

20. W. Swartout, Y. Gil, A. Valente (1999): "Representing Capabilities of Problem Solving Methods". *Proc. IJCAI-99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends*. Stockholm, Sweden, (1999).
21. Vetter, J., Schwan, K.: "Techniques for High-Performance Computational Steering". *IEEE Concurrency* **7**(4), October-December (1999). 65
22. Wickler, G.: "CDL: An Expressive and Flexible Capability Representation for Brokering". *Proc. ICMAS 2000*, to appear (2000). 69
23. Zapf, M., Geihs, K.: "What Type Is It? A Type System for Mobile Agents". In Trappl, R. (ed.) *Cybernetics and Systems 2000*, "Osterreichische Studiengesellschaft f"ur Kybernetik, Wien, pp. 585–590 (2000). 74

Models of Coordination

Robert Tolksdorf

Technische Universität Berlin, Fachbereich Informatik, FLP/KIT
Sekr. FR 6–10, Franklinstr. 28/29, D-10587 Berlin, Germany
mailto:tolk@cs.tu-berlin.de

Abstract. While future software is becoming decomposed in more and more finegrained entities, issues on interactions amongst those entities grows in importance. While methodologies for building such components are well established, the design and support of their interplay can not build on commonly understood and well defined models.
In this paper, we review several coordination models from various disciplines, and describe how a coordination reference model could look like. We use a set of characteristics of coordination models to compare the reviewed ones.

1 Looking at Models of Coordination

Todays software is structured into modules, objects, components, agents etc. These entities try to capture rather small conceptual abstractions and supports it with functionality. While this is advantageous for the design and implementation of software, networked environments add additional benefits when running programs composed from those entities. Given a good encapsulation, they can be distributed or mobile, and different non-functional characteristics such as fault-tolerant or persistence can be attributed to them.

And in fact, the tremendous advances in network technology in terms of availability, cost and functions, has impact on how software is composed. Brereton *et al.* [3] state: "Software will be fine-grained. Future software will be structured in small simple units that cooperate through rich communication structures and information gathering."

While the implementation of these small units is is commonly well understood and methodologies for their design become more and more adopted, the question on how the cooperation amongst them is designed and supported. Kahn *et al.* [11] point out this change of focus: "The overall applications challenge, from a processing and communications viewpoint, is how to implement complex 'ensemble' behavior from a large number of individual, relatively small sensors."

Todays technologic state-of-the-art gives a first impression of the size of the challenge. With small devices being enhanced by processing power and its miniaturization one can expect, that within the next few year, the number of units coordinating will increase by several magnitudes. For example, a GPS receiver device was announced in [19]: "SiRFstarII will enhance location accuracy by an order of magnitude – from 100 meters to between 2 and 15 meters. SiRF

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 78–92, 2000.

has included a 32-bit, full-function, widely supported CPU onto the SiRFstarIIe chipset [...]. The CPU is a 32-bit, 50 MHz ARM processor, with fully 90 percent of the throughput available for non-GPS tasks." The chipset mentioned has the size of a US-quarter coin.

Taken to an extreme, miniaturization will lead to computing devices that are very small in size and that will be very cheap to produce in large numbers. They have certain characteristics, that lead to what is described in [1] as "[...] the challenge of amorphous computing: How does one engineer prespecified, coherent behavior from the cooperation of immense numbers of unreliable parts that are interconnected in unknown, irregular, and time-varying ways?"

It is the interaction of computing units that makes their composition useful. Enabling for a useful interaction is coordinated activity. And in order to provide technology that supports the interaction and its design, models of coordination are necessary. These models have to have certain qualities such as being complete with respect to interaction forms and open to new patterns of interactions. They have to be easy and safe to use to facilitate efficient software engineering. They must be scalable and efficient to implement to cope with the number of units to coordinate. And finally, the models have to be aware of the characteristics of future environments, eg. be robust to failures and dynamics.

Coordination models can be used to build middleware and coordination languages. Given a set of coordination primitives, they can be used to express coordination strategies that lead to the mentioned coherent behavior of interacting entities. Thus coordination models are enabling for the execution and design of coordinated applications. If it is possible to find commonalities amongst coordination models, then coordination patterns used in the various disciplines could be made transferable.

In this paper we take a first step towards that goal and look at a variety of existing coordination models from different sources. We evaluate them with respect to a set of qualities and ask how a coordination reference model – the meta-model of the coordination models reviewed – could look like.

2 Sources of Coordination Models

Models of coordination are widespread and come in a variety. A large set of domains and disciplines has developed their own models on how entities interact, and do in part also provide technologies that take advantage of these models. Examples are:

- Daily life naturally suggests that independent entities cooperate. Thus, *everyone* has some – diffuse – understanding of what coordination is and should at least be able to tell uncoordinated behaviors from coordinated ones.
- In *computer science*, parallel programming has been one of the first fields in which coordination models were developed. The fields of distributed computing has added further models [4].
- *Distributed AI* is concerned with the design of coordination in groups of agents and uses a variety of models.

- *Organization theorists* try to predict the future behavior and performance of organizations. As in these the interaction of actors is a key factor, coordination models are used.
- *Economics* looks at how interactions in markets takes place and models them.
- *Sociologists* and *Psychologists* try to explain the behavior of groups composed by individuals [9].
- *Biologists* study natural phenomena in natural agent systems such as ant colonies or swarms and try to discover the embodied coordination mechanisms [2].

This incomplete list already indicates that there is a huge variety of coordination models. But there is currently no consensus on the relations between coordination, communication and cooperation. Although these individual models each are able to explain important characteristics of collaboration, coordination and communication, the definitions used are often in conflict.

In order to compare and integrate such models, it is necessary to work towards a standardized terminology which contains terminological definitions and clarifications of basic notions including the very term "coordination". Based on that, a conceptual model has to relate those terms in the most general and flexible manner. With such building blocks, a uniform representation of collections of coordination patterns would be enabled and the patterns could be implemented coordination services. In [12] these four dimensions are considered a road map towards a reference model for communication, coordination, and cooperation.

3 Models of Coordination

In this section we examine several coordination models. We begin with the naive view on coordination and then examine two models coming from an organizational perspective, namely the model on organizational structure by Mintzberg and the coordination theory model from Malone and Crowston. We briefly look at formal models and distributed artificial intelligence. Finally, we examine two views on coordination from computer science, the Linda model by Gelernter and Carriero and the workflow reference model by the Workflow Management Coalition.

3.1 Naive Model

A common understanding of coordination is that active entities coordinate to achieve a common goal.This understanding assumes a common goal shared by all entities. This goal can be explicitly stated and represented, but can also be implicit. The entities can be assumed to be willing to cooperate, that is to follow the goal. They can do so explicitly by cooperating, but this is not required.

The naive model provides no clear focus on coordination. Coordination is not encapsulated outside the agents. Coordination mechanisms and policies are therefore not interchangeable.

3.2 Mintzberg Model

[16] is a seminal work by Mintzberg on structures of organizations, coordination mechanisms used therein and dominant classes of configurations of organizations.

Mintzberg develops a theory on the structure of organizations by postulating five basics parts which can be found in any organization. As depicted in figure 1, at the basis of the organization is the *operating core* where the actual work is performed. At the (hierarchical) top of organizations is the *strategic apex* – managers who have the overall responsibility for the organization and that take strategic decisions and guide the direction of the organization. The *middle line* is a chain of managers that implement the decisions by supervising subordinates and reporting to their supervisors. The *technostructure* serves to analyse and organize the work done. *Support staff* includes all the indirect support of work, eg. by running a plant cafeteria.

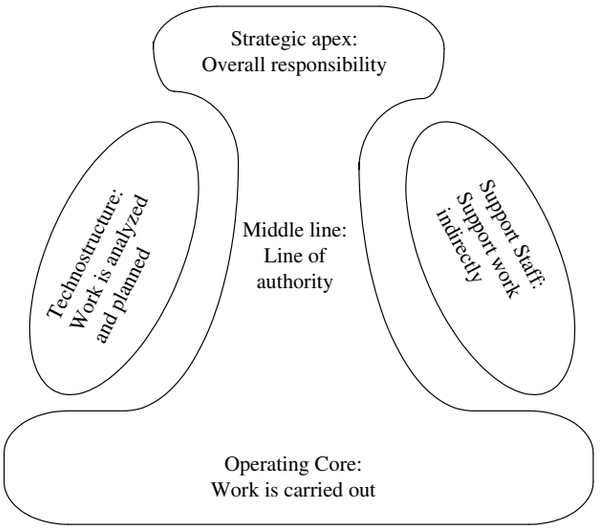
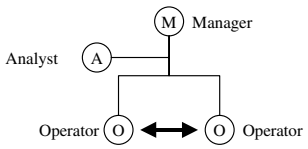
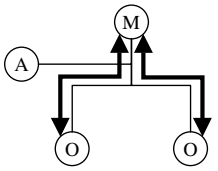
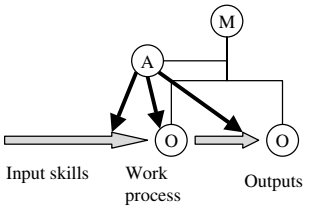


Fig. 1. The structure or organizations

Coordination in organizations is explained by five mechanisms as shown in table 1:

1. *Mutual adjustment* builds on informal communication amongst peers. There is no outside control on decisions and peer coordinate their work themselves.
2. With *Direct supervision*, a supervisor coordinates the work of its subordinates by giving instructions.
3. *Standardization of work* ensures coordination by specifying the work to be done so that no decisions have to be taken later.

Table 1. The five coordination mechanisms in organizations [16]

Mutual adjustment: Informal communication	
Direct supervision: One is responsible for others	
Standardization of work Standardization of outputs Standardization of skills	

4. *Standardization of outputs* refers to specifying the result of work, thus these can be used by others without additional coordination.
5. *Standardization of skills* specifies the training necessary for a specific work. Additional coordination then is not needed, as peers know what to expect from each other.

From the resulting design space for organizations, Mintzberg selects five configurations of coordination mechanism and preeminent part in the organization as in table 2.

Table 2. The five structural configurations of organizations [16]

Name	Coordination mechanism	Key part
Simple structure	Direct supervision	Strategic apex
Machine Bureaucracy	Standardization of work processes	Technostructure
Professional bureaucracy	Standardization of skills	Operating core
Divisionalized form	Standardization of outputs	Middle line
Adhocracy	Mutual adjustment	Support staff

The Mintzberg model assumes a role model for actors in an organization. The choice of coordination mechanisms is induced by the choice of the organizations structure and thus not easily exchangeable. Actors are very aware of the coordination mechanism they have to use. Mintzberg discusses patterns of transitions amongst the different configurations, but these tend to be rather slow.

3.3 The Coordination Theory Model

Malone and Crowston [14] introduce the term *coordination theory* to "refer to theories about how coordination can occur in diverse kinds of systems." It draws on different disciplines such as computer science, organization theory, management science and others.

The model defines coordination as *the management of dependencies amongst activities*. In order to make an interdisciplinary use of coordination mechanisms found in various kinds of systems, the processes involved in the management of dependencies have to be studied.

In order to do so, the kinds of dependencies have to be analyzed and the respective processes be studied [6,7].

Table 3. Coordination mechanisms and managed dependencies

Coordination mechanism	Dependency managed	
Resource allocation		Shared resources
Notification Transportation Standardization	Prerequisite Transfer Usability	Producer/ Consumer
Synchronization		Simultaneity
Goal selection Decomposition		Task/Subtask

The model assumes that the dependencies are external to activities studied. It is not the activities that are managed, but relations amongst them. The model abstracts from the entities that perform activities and their goals.

3.4 Formal Models

In this subsection, we briefly look at formal models of coordination. We follow the overview in [17] and refer to this source for a closer description.

In centralized formal models, there is a known global set of entities to be coordinated. The state of each with respect to coordination activities is modeled by a decision variable. Thus, the system to be coordinated is represented by a set of decision variables $V = \{v_1, \ldots, v_n\}$ (with values from a set of domains $D = \{D_1, \ldots, D_n\}$). Any coordination process leads to an instantiation x of decision variables from decision space X.

In quantitative formal models, a global utility function $U : X \rightarrow \Re$ is associated with each of these instantiations that models how good the system is coordinated. The model can be analyzed to find an optimum $y \in X$ such that $\forall x \in X : U(x) \leq U(y)$, meaning that there is no instantiation that provides better coordination.

For a qualitative mode, a set D of constraints $\{C_1, \ldots, C_m\}$ is used. The notion of a consistent instantiation x of decision variables from decision space is defined by X as $y \vDash C_1 \wedge \ldots \wedge C_m$.

Game theory provides a model of decentralized cooperation. The set of entities to be coordinated is modeled as a game which consists of a set I of n players. There is a space S of joint strategies $S = S_1 \times \ldots \times S_n$. It collects the individual strategy $S_i = \sigma_{i_1}, \ldots, \sigma_{i_m}$ that each player has.

In contrast to the global utility function of the quantitative model above, a set P of payoff functions is defined for each player individually by $P_i : S \rightarrow \Re$.

Two kinds of games are distinguished. In zero-sum games the payoff of one player is "financed" by lower payoffs of the others: $\forall \sigma \in S : \sum_{i=1}^{n} P_i(\sigma) = 0$. In non-constant sum games, this restriction does not exist: $\exists \sigma, \sigma' \in S : \sum_{i=1}^{n} P_i(\sigma) \neq \sum_{i=1}^{n} P_i(\sigma')$.

The situation can be analyzed in two ways. In non-cooperative analysis, the players try to get the best payoff they can individually. The set of strategies is said to be in a Nash equilibrium, if deviation from it by one player will not increase that players payoff: $\forall i \in I : \forall \sigma_i \in S : P_i(\sigma_1^*, \ldots, \sigma_i, \ldots, \sigma_n^*) \leq P_i(\sigma_1^*, \ldots, \sigma_i^*, \ldots, \sigma_n^*)$. In a cooperative analysis, the players coordinate strategies and join payoffs. The situation is said to be Pareto-optimal if no one can achieve a higher payoff without lowering that of some other player.

All these formal models share some assumption. First, they make a fundamental assumption about the environment, namely that payoff and utility can be defined exact and static. Second, they assume that agents behave exactly rational to maximize utility or payoff.

3.5 Coordination Mechanisms in DAI/MAS

Jennings [10] asserts that the key to understanding coordination processes to to look at the internal structures of agents. There, commitments – pledges of agents about actions and beliefs in the future or the past – and conventions – general policies on reconsiderations of commitments – are determent for coordination mechanisms. In addition, social conventions give policies on interactions in a community of agents and local reasoning is necessary to use that knowledge.

In Computational Organization Theory, roles are defined that constrain behavior of agents. In Multi-agent Planning, commitments are based on plans that agents develop. In a multi agent society setting, negotiation is the coordination mechanism by which agents take joint decisions after following some negotiation protocol.

The models assume that the conventions governing the coordination processes are external to the agents. They assume commitments a priori to events

that take place, and thus assume knowledge about future events. Also, agents act rational.

3.6 Uncoupled Coordination

In parallel computing, questions on how to organize the execution of multiple concurrent threads in a computation has led to several coordination models. The model embodied in the language Linda [8] takes the view that coordination has to be performed explicitly by the parallel processes and that it is worthwhile to use a separate language for that. Such a coordination language focuses merely on the expression of coordination and defines a respective coordination medium [5].

The language Linda embodies a model of uncoupled coordination where a set of agents together form an ensemble in which they coordinate their interaction indirectly by using a shared dataspace, called the *tuplespace*.

The coordination language is defined in terms of operations that access and modify the tuplespace. The operation *void out(Tuple t)* emits a piece of data – a *tuple* which is a list of values of some primitive data types – into the tuplespace. To retrieve it, one uses the operation *Tuple in(Template t)*, which takes a *template* of a tuple to describe what kind of tuple is sought. A template can contain actual values or placeholders denoting only the type of a value expected in some field of the tuple. The operation blocks until a tuple that matches the template is emitted, removes that tuple from the tuplespace and returns it to the agent that issued the *in*. *Tuple rd(Template t)* basically does the same, but leaves a copy of the matching tuple the space.

The model assumes explicit coordination amongst agents that have to use the coordination language in an appropriate way. The pattern of coordination is scattered over the use of the coordination operations with the agents.

The model provides an abstraction from the location of agents in space and time. Agents remain anonymous to one other and do not necessarily have to exist at the same time. Also, it abstracts from the computational model and programming language used by the agents.

3.7 Workflow

Workflow Management Systems (WfMS) coordinate human work and its support by applications. In recent years, there has been an urge to define a common denominator of such workflow modeling languages to enhance interoperability amongst systems of different vendors. The Workflow Management Coalition (WfMC) is the industry consortium of the leading WfMS vendors and has published a reference model. Part of it is a process definition language [21] that represents a minimal language to express workflow models.

Here, a workflow is modeled as a graph of activities as nodes and transitions between them. The transitions represent dependencies amongst activities and can be augmented with additional constrains, such as start- and end-times. The topology of the graph includes coordination constraints on activities.

So called AND-JOIN- and AND-SPLIT-nodes synchronize activities or span new parallel activities. An XOR-JOIN makes the execution of an activity dependent on the termination of one out of several other ones. XOR-SPLIT selects one new thread of activities to start. The flow of activities can be further specified by introducing loops and sub-workflows.

As shown in figure 2, activities are performed by participants in the workflow and might involve data and applications. The participants are constituent for the organization in which the workflow takes place.

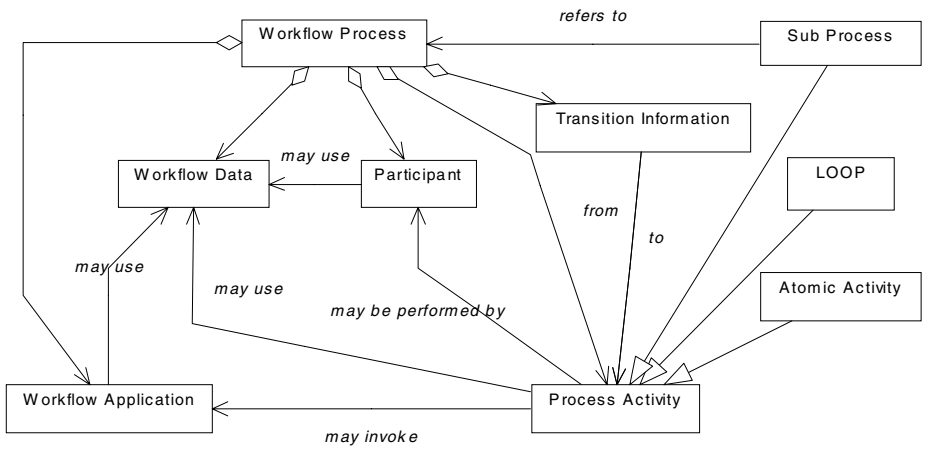


Fig. 2. The reference model of the WfMC

The approach taken by most WfMS assumes that all activities and dependencies amongst them are known in advance. Also, reliable execution of activities is assumed – at least the reference model lacks the notion of exceptions.

4 Towards a Coordination Reference Model

In this section, we outline how a coordination reference model could look like, define attributes for coordination models and review the models from the preceding section with respect to those.

4.1 Structure of a Coordination Reference Model

We propose the following constituents for a coordination reference model. As shown in figure 3, four model layers (see [13]) seem of interest to us.

While the real world knows situations in which coordination is neither present nor necessary, we are interested only in that part of the world in which activities

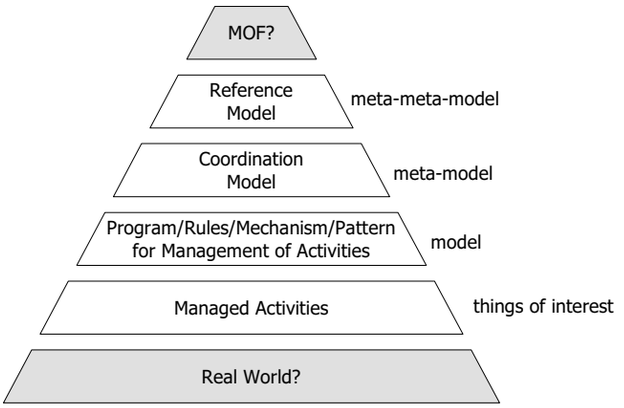


Fig. 3. A structure for a coordination reference model

are managed in order to be coordinated. These form the object-level in our model-hierarchy.

For a specific set of such objects, the concrete management of activities is described by a set of rules, specific mechanisms, programs or a selection of coordination patterns. Thus, they are models of specific managed activities – the blueprints for actual interactions.

The coordination models described in the preceding section each provide a specific framework to express such models. Thus, they are meta-models above the specific models that describe specific managements of sets of activities.

The coordination reference model that we are interested is a meta-model to the coordination models. It contains terminologies and concepts to describe coordination models.

The reference model itself also has a meta-model that describes for example, what a concept or a term is. For our purposes, we are not interested in that model layers, and would make use of some existing meta-model, such as the OMG MOF.

Within the reference model, the following set of concepts seems necessary:

- *Interactors* are those entities that are related to other interactors.
- *Relations* associate two or more interactors in some way. Coordination mechanisms then apply to relations amongst interactors.
- *Non-Interactors* are those entities that have no relations to interactors.
- *Operations* can be performed by interactors on non-interactors.
- *Attributes* can be assigned to interactors and non-interactors do describe them or their current state.
- *Meta-Attributes* contain the characteristics that describe the models built from the preceeding five concepts.

As an example of how these concepts describe a coordination model, we can look at a part of the Mintzberg model from organization theory.

- *Manager*, *Line-Manager*, *Analyst*, *Operator*, *Support staff* are instances of *Interactor*.
- *controls* is a *Relation* put forth by the model and relates a manager with an operator. There is no further coordination mechanism described than the enactment of that relation.
- *input*, *output*, *skills* are *Non-Interactors*.
- *Learn* is an *operation* by which an operator can increase his/her skill.
- *Span of control* is an *attribute* of a line-manager.

4.2 Comparing Coordination Models

We believe that all the models reviewed can be described by the above reference model, which is enabling for their comparison. In order to compare them, we now introduce a set of meta-attributes of the models. Highlighted are those characteristics that we consider dominant for a model.

- To what degree the model provides a *clear distinction* amongst interactors, non-interactors and management of relations.
- How *orthogonal* the coordination model is with computational models used by interactors.
- The degree of *coupling* between interactors. This also includes coupling to an external goal (modeled as a non-interactor).
- The degree of *autonomy* of interactors. It is inverse to the degree of centralization assumed or introduced by the model.
- Whether the management of relation is *external* to interactors or not.
- How much *awareness* to management of relation is required from interactors.
- The degree of *stability of interactors* assumed by the model on the interactors, eg. whether there is a static set of them or not.
- The *stability of relations* assumed by the model, ie. long- vs. short-term relations. Longterm means the lifetime of the situation in which coordination is necessary. Midterm means the lifetime of one interaction and shortterm refers to single coordination activities within an interaction.
- How much *reliability* is assumed about the interactors
- The *scalability* provided by the model with respect to the number of interactors and relations amongst them.
- How *usable for programming* a coordination model is. A model suited for prescriptive modelling – as opposed to descriptive modelling – can be used as programming model.
- Whether there are qualitative or quantitative *measures* on the management of relations provided.

Further dimensions seem useful but have not been considered here. Examples are realtime aspects or how a model views the environment of interactors. With respect to those dimensions listed, table 4 shows a comparison of the models reviewed.

Model	Distinction	Orthogonal	Coupling	Autonomy	External	Awareness	Interactors stability	Relations stability	Reliability	Scalability	Programming	Measures
Naive	No	Yes	High	**Low**: Shared goal	No: Entities coordinate	Yes	Midterm: Until goal reached	Midterm: Until goal reached	Yes	No	No	No
Mintzberg	Yes: Inputs and outputs in addition to actors	Yes	Dep. on mechanism	Low	No	Yes	Midterm	**Longterm**	No	Yes	Unclear	No
Coord. Theory	Yes	Yes	**Dep. on mechanism**	Mid	Yes	Yes	Midterm	Midterm: Until dissolved	Yes	No	Unclear	No
Formal	No	**No**: Evaluation of functions in model	High	Low	No	Yes	Longterm	Longterm	Yes	No	Yes	**Yes**
DAI/MAS	Yes	Yes	Mid	High	Yes	**Yes**	Midterm	Midterm: Horizon of reasoning	No	**No**	Yes: Agent systems	No
Linda	Yes	Yes	**Low**	Mid	Yes	Mid	**Shortterm**	**Shortterm**	Yes (see [20])	Unclear (see [15])	Yes	No
WfMC	Yes	Yes	High	Low	Yes	Yes	**Longterm**	**Longterm**	Yes	No	Yes	Yes

Table 4. Comparing the coordination models

Some of the entries are highlighted and are explained as follows. Formal models provide poor orthogonality as the utility and payoff functions are calculated as part of the internal workings of the agents coordinated.

The Minzberg model is hard to asses with respect to the degree of coupling. On the one hand, the actors within an organization can be uncoupled if the coordination mechanism applies allows for it. On the other hand, all actors in an organization are tightly coupled by forming that very organization. Linda incorporates an extremly uncoupled style of coordination, as agents need not exist at the same time and place in order to exchange tuples.

The autonomy of agents in the naive model is rather low, as they follow a common goal. DAI/MAS agents are aware of coordination if the agents form models of other agents.

The stability of interactors required in Linda is extremly shortterm – the agents do not even have to exist during the whole interaction. Workflow models are at another extreme: Workflow schemas are commonly static and assume a fixed set of interactors.

The stability of relations amongst interactors is equally shortterm with the Linda model. The Minzberg model assumes rather longterm structures and only few changes within the organizations lifetime. Comparable, static workflow models assume static relations.

The scalability of DAI/MAS is limited if reasoning is involved. Most recently, however, scalable models are emerging (see [18,2]).

For the naive model, the existence of a shared goal is the dominant characteristic. It induced a high coupling and determined the stability of interactors and relations.

The Mintzberg model is characterized by the assumed stability of the relations, which matches steps in the life-cycle of an organization. The model shows a high flexibility as it offers multiple coordination mechanisms The coordination theory model knows about an extensive set of possible coordination mechanisms and models their relation towards dependencies to be managed. This is the dominant characteristic of this model. Stability of dependencies is assumed to be midterm, as one interaction dissolves a dependency.

Formal models provide at their core measures of how optimal a situation is coordinated. They tend to be centralized and thus are not well scalable.

DAI/MAS models assume reasoning of agents about their own coordination activities, thus they are highly aware of coordination. However, this also leads to scalability problems. The Linda model is characterized by low coupling of interactors in space and time which imposes low requirements of stability of interactors and relations. Dominant for workflow models is the assumption that interactors and relations – and thus coordination procedures – are stable of a long time.

The overview thus shows that each model can described by dominant characteristics along our dimensions. Also, it shows that the models reviewed here cover very well the scale of those dimensions.

5 Conclusion

In conclusion, we have seen that the variety of coordination models is large and draws on various disciplines. After reviewing them, we have tried to outline a reference model of coordination which could be capable of serving as a meta-model to those reviewed.

With respect to to a set of characteristics of models, we found that the models are well-distinguishable along those dimensions. We found that each model has a dominant characteristic. We also found that the set of models covers substantial parts on the scales of the dimensions considered.

The author would like to acknowledge the anonymous reviews for their most valuable comments.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous computing. Technical Report AI Memo 1665, MIT AI Lab, 1999. 79
2. Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence*. Oxford University Press, 1999. 80, 90
3. Pearl Brereton, David Budgen, Keith Bennnett, Malcolm Munro, Paul Layzell, Linda MaCaulay, David Griffiths, and Charles Stannett. The future of software. *Communications of the ACM*, 42(12):78–84, December 1999. 78
4. Roger S. Chin and Samuel T. Chanson. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):91–124, March 1991. 79
5. P. Ciancarini. Coordination Models and Languages as Software Integrators. *ACM Computing Surveys*, 28(2):300–302, 1996. 85
6. Kevin Ghen Crowston. *Towards a Coordination Cookbook: Recipes for Multi-Agent Action*. PhD thesis, Sloan School of Management, MIT, 1991. CCS TR# 128. 83
7. Chrysantos Nicholas Dellarocas. *A Coordination Perspective on Software Architecture: Towards a Design Handbook for Integrating Software Components*. PhD thesis, Massachusetts Institute of Technology, 1996. 83
8. David Gelernter and Nicholas Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992. 85
9. Jonathon Gillette and Marion McCollom, editors. *Groups in Context, A New Perspective on Group Dynamics*. University Press of America, 1995. 80
10. Nicholas Jennings. Coordination techniques for distributed artificial intelligence. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 187–210. John Wiley & Sons, 1996. 84
11. Kahn, Katz, and Pister. Next century challenges: Mobile networking for "smart dust". In *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*, 1999. 78
12. Matthias Klusch, Paolo Petta, Dieter Fensel, and Jeremy Pitt. Premises and challenges of research and development in information agent technology in europe, 1999. Technological Roadmap of the Special Interest Group on Intelligent Information Agents as part of the ESPRIT Network of Excellence for Agent-Based Computing. 80

13. Cris Kobryn. UML 2001: a standardization odyssey. *Communications of the ACM*, 42(10):29–37, October 1999. 86
14. Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994. 83
15. R. Menezes, R. Tolksdorf, and A. M. Wood. Scalability in LINDA-like coordination systems. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer, 2000. 89
16. H. Mintzberg. *The Structuring of Organizations: A Synthesis of the Research*. Prentice Hall, Englewood Cliffs, N. J., 1979. 81, 82
17. Sascha Ossowski. *Co-ordination in artificial agent societies: social structures and its implications for autonomous problem-solving agents*, volume 1535 of *LNCS*. Springer Verlag, 1999. 83
18. H. V. D. Parunak. "go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997. 90
19. SiRF Technology, Inc. Gps goes mainstream. Press release, August 1999. http://www.sirf.com. 78
20. R. Tolksdorf and A. Rowstron. Evaluating fault tolerance methods for large-scale linda-like systems. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, 2000. 89
21. Workflow Management Coalition. Interface 1: Process definition interchange process model, 1998. http://www.wfmc.org. 85

From Analysis to Deployment: A Multi-agent Platform Survey

Pierre-Michel Ricordel and Yves Demazeau[1]

LEIBNIZ laboratory,
46 av. Felix Viallet, 38000 Grenoble, France
{Pierre-Michel.Ricordel,Yves.Demazeau}@imag.fr

Abstract. This paper presents a survey on multi-agent platforms, with a particular focus on methodology. It presents the four stages of the construction of a multi-agent system and derives from these stages several criteria for comparison. These criteria are then used to evaluate four selected platforms, and lead to a discussion on the future of multi-agent platforms.

1 Introduction

Recently, natural evolution of Multi-Agent Systems (MAS) has leaded them to migrate from the research laboratories to the software industry. This migration is good news for the entire MAS community, but it also leads to new expectations and new questions about multi-agent system methodologies, tools, platforms, reuse, specification, and so on. This introduction of Software Engineering techniques into Multi-Agent Systems gains more and more interest in both the research area and the industry. The best example is the brunch of announcements about new multi-agent platforms, usually supposed to be the ultimate tool to build multi-agent systems. Since this kind of affirmation seems far-fetched compared to the difficulty of the problem, we need some elements to evaluate and to compare multi-agent platforms. In this paper, we propose an analysis grid, built from criteria that evaluate each of the stages encountered during the creation of multi-agent systems. Of course, as in benchmarks, any criteria is relevant to a specific outside need, and a platform can only be compared relatively to another one because there is nothing like an absolute measurement scale for multi-agent platforms. But some advance can be made in platform comparison by proposing comparison criteria and applying them to multi-agent platforms as in this paper. In section 2 we present the MAS construction stages, then, in section 3, the evaluation criteria drawn from these stages. In section 4 we evaluate four multi-agent platforms (AgentBuilder, Jack, MadKit, Zeus) with the established criteria, and finally we discuss the results in section 5. Since the multi-agent platform market is evolving very fast, we would like to insist on the fact that the platforms analysed in this paper

[1] Yves Demazeau is research fellow at CNRS (Centre National de la Recherche Scientifique).

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 93-105, 2000.

are evaluated given their current state at the time of writing this paper (first semester 2000), and that their specifications may change as time goes on.

2 Four Stages to Build Complex Software

Multi-Agent Systems are complex software. Building such complex systems software requires using adequate engineering methods. Traditionally, software engineering distinguishes three stages of construction: Analysis, Design and Development. We believe that for systems like Multi-Agent Systems this is not sufficient, and that describing the entire MAS creation process, from early design to a running application, we have to distinguish a fourth stage after the development stage, called deployment.

We also know that the exact frontiers between the three classic stages (analysis, design and development) are still subject to debate in software engineering. Since we believe that these stages are just quantified levels along a natural conceptual continuum, we propose to use the following definitions for these four stages:

- Analysis: the process of discovering, separating and describing the type of problem and the surrounding domain. Practically, it consists of identifying the application domain and the key problem.
- Design: the process of defining the solution architecture of the problem, in a declarative way. Practically it consists of specifying a solution principle of the problem, for example using UML [1].
- Development: the process of constructing a functional solution to the problem. Practically it consists of coding the solution with a particular programming language.
- Deployment: the process of effecting the solution to the real problem in the given domain. Practically it consists of launching the software on a network of computers, and then, to maintain or to extend its functionality.

The two former stages are more involved with methodology and the two latter addresses more technical aspects. In this paper we focus on the different methodological stages, but the software development process support is also an important issue.

3 Criteria of Examination

In this part we will present the different criteria we have drawn from the development stages presented before.

3.1 Four Qualities of Development Stages

The process of drawing common *qualities* from the four building stages is not evident. By *quality*, we mean a distinct feature that characterises a positive or a negative aspect

in the practical realisation of a particular stage. We have determined four qualities that seem relevant to us to all the stages, and which address as many aspects as possible of practical development pitfalls. These are:

- Completeness: The degree of coverage the platform provides for this stage. This addresses both the quantity and the quality of the documentation and tools provided.
- Applicability: The scope of the proposed stage, in other words the range of possibilities offered and the restrictions imposed by the proposed stage.
- Complexity: The difficulty to complete the stage. This includes both the competence required from the developer and the quantity of work the task requires.
- Reusability: The quantity of work gained by reusing previous works.

In the next section, we will apply these qualities to the four stages of development.

3.2 Application of the Qualities to the Stages

The four qualities applied to the four stages of development result into sixteen criteria for evaluating any platform.

Analysis The Analysis criteria includes:
- Completeness: is the analysis method useful, and is it well documented?
- Applicability: to which domains and problems does this method apply?
- Complexity: is the analysis method easy to understand and easy to apply?
- Reusability: is there a way to exploit previous analysis of similar problems or of similar domains? Are there analysis examples supplied?

Design The Design criteria includes:
- Completeness: is the design method useful and well documented? Are there tools to support the design process?
- Applicability: what kind of MAS can be designed by this method?
- Complexity: is the design method easy to understand and to apply?
- Reusability: is there a way to reuse existing designs? Are there useful designs supplied?

Development The Development criteria includes:
- Completeness: How useful are the supplied development tools?
- Applicability: Is there some functionality impossible to achieve with the development tools?
- Complexity: Are the development tools and languages easy to use? How popular is the language used?
- Reusability: Is there an active support to reuse the code?

Deployment The Deployment criteria includes:
- Completeness: Is there a support for the deployment of the MAS?
- Applicability: Does the deployment tool support visualisation, profiling, on-line maintenance, etc?
- Complexity: Are the deployment tools easy to use and understand?
- Reusability: Is it possible to integrate a previously existing agent dynamically into a new multi-agent system without any modification?

3.3 Other Criteria

We can also consider some other practical criteria that one has to take into account when choosing a platform. In particular, these criteria can condition the adoption of a platform to a particular project:

- Availability: Is there a trial version, confidential clauses, is the source code available? How much does it cost?
- Support: What are the future developments of this platform? Is this platform used in large scale?

At a finer level, we could explore more technical aspects, such as process handling, mobility, security, standardisation, but we limit our analysis to a more conceptual level.

4 Platform Examination

4.1 Choice of the Platforms

We have chosen a set of platforms that have in common:
- to be popular and regularly maintained (for bug fixes and extra features),
- to be grounded on well-known academic models,
- to be developed with industry-like quality standards,
- to cover as many aspects as possible of Multi-Agent Systems, including agent models, interaction, coordination, organisation, etc.,
- to be simple to set-up and to evaluate. This includes good documentation, download availability, simple installation procedure, and multi-platform support.

We have deliberately avoided platforms that:
- are still in experimental state, abandoned, or confidentially distributed,
- are grounded on vague or non-existent models,
- cover only one aspect of multi-agent systems, like single agent platforms, mobile agent platforms, interaction infrastructures toolkits,
- Are too short on some construction stages, like purely methodological models or development tools without methodology.

We would like to lay stress on the fact that in this study we have deliberately chosen platforms coming from the Multi-Agent community towards Software Engineering, and not platform coming from the Object-Oriented, Component-Oriented or Software Engineering community towards Agent systems. This includes popular mobile agent platforms, such as Voyager, Grasshopper and others. This kind of platform has very good deployment tools, but generally does not address at all Multi-Agent specificities, especially at the analysis and development stages.
Given these criteria, we have selected four platforms to be presented in this paper:

- AgentBuilder®
- Jack™
- MadKit
- Zeus

4.2 Commented Evaluation

In this section, we evaluate the four selected platforms, using the above mentioned criteria, each platform is followed by a short comment.

AgentBuilder® (`http://www.agentbuilder.com/`)
AgentBuilder is an integrated tool suite for constructing intelligent software agents. It is developed by Reticular Systems Inc., and is grounded on the Agent0 [2][3] and Placa [4] BDI models. This tool is remarkable both by the high quality of its software and the well-known academic background model used. The global methodology is shortly described in the AgentBuilder User's Guide [5].

Analysis The analysis stage consists of the specification (in OMT) of the objects in the domain and the operations they can perform, followed by the production of a domain ontology. Graphical software tools support this stage, but very few hints are given about how to perform a good analysis. This kind of domain decomposition is almost applicable to any domain/problem. To complete this stage, a good experience in object modelling is sufficient, and the provided tools are quite intuitive to use. On the reusability aspect, AgentBuilder supports the reuse of ontologies through an ontology repository.

Design The design stage consists of the decomposition of the problem into functions that agents may perform. Then the agents are identified, their roles and characteristics are defined and finally the interaction protocols that they use are designed. Here also, efficient software tools (the Agency Manager and the Protocol Editor) are provided, but few indications are available to achieve a good design. This stage is applicable to the design of any multi-agent system composed of intelligent agents. In order to complete this stage, a solid background in multi-agent design is preferable, but here also, the tools are easy to use. The reuse of protocols is possible with the protocol repository.

Development The development stage consists of defining the behaviour of agents, more precisely the behavioural rules, initial beliefs, commitments, intentions and capabilities of agents. It is also during this stage that external actions and agent Graphical User Interfaces are integrated into agents, using the Project Accessory Class library. Graphical software tools support all these tasks. This stage is only applicable to agents that use the proposed BDI architecture. A background in logic programming and BDI models is needed, but the proposed BDI model is a classical one, and the graphical tools simplify the construction of behavioural rules, avoiding syntactic and semantic errors during their construction. Finally, agents and external actions can be reused across projects.

Deployment A Run-Time Agent Engine, that interprets agent programs, executes every agent generated during the previous stages. A debugging tool is also provided, allowing to monitor the agents' mental model step-by-step, and to monitor agent's interactions. Due to the full BDI agent model, only MAS composed of a few cognitive agents are achievable. The complexity of deployment is reduced by graphical and intuitive tools.
Agents may dynamically be added to a running multi-agent system, but this feature is not directly supported by the tools.

Other AgentBuilder is a closed-source commercial product. A free evaluation version is available, limited to the tutorial agents. It is available in three versions: AgentBuilder Lite, AgentBuilder Pro and AgentBuilder Enterprise. Academic versions are also available. At the time of writing this paper, license fees range from $100 to $5000 US$ depending on the version. Phone and electronic support is available. The product is still under evolution.

Comments AgentBuilder's documentation covers almost all the stages, from analysis to development. This is a good point, even if the analysis and design parts are quite succinct. They deal more with *what to do* rather than with *how to do*, which is more difficult to determine but is more helpful to the developer. Another good point is the software tool, which cover almost all the aspects of the stages, and establish links between them (as the automatic generation of behavioural rules from protocol definitions). The disadvantage of such a frame is that it limits the versatility of the tool. Multi-Agent Systems constructed with AgentBuilder are homogeneously composed of agents that use the AgentBuilder agent model. There is no easy way with AgentBuilder to integrate agents that use another model, or to interact with an environment. As any complex tool, AgentBuilder is long and difficult to learn, but once the tool mastered, can become very productive.

Jack™ (`http://www.agent-software.com.au/`)
Jack is described as *an environment for building, running and integrating commercial JAVA-based multi-agent systems using a component-based approach*. It is developed by Agent Oriented Software Pty. Ltd., an Australian commercial company. It is based on the BDI model dMARS developed at the Australian Artificial Intelligence Institute

(AAII) [6]. Jack focuses mainly on the development stage. Analysis and design steps are only mentioned in [7]. The toolkit consists of the JDE (Jack Development Environment), a graphical tool to manage projects, the Jack Agent Language (JAL) compiler, that translates JAL programs to pure Java programs, and a library of supporting classes, called the Jack Agent Kernel.

Analysis There is no evidence about any analysis method.

Design It is assumed that the definition of the agents' functionality has been provided, and that the BDI model has been chosen. Then, this stage consists of the identification of the elementary classes required to manipulate the domain objects (perceptions, actions, domain-specific data structures), and the identification of the mental states elements of the agents (interactions, goals, beliefs, plans). This applies only if you choose the proposed Jack's BDI agent model. In this case, a good knowledge of the Jack BDI model is required. The lack of a real design method makes reuse difficult.

Development Development is a combination of normal Java coding for the elementary classes and extended Java (Jack Agent Language) for the agent-specific components. The extensions allow describing agent's behaviour, capability, plans, events and relational database into Java code. The Jack Agent Compiler then translates this extended Java into pure Java. A graphical software tool is also provided to facilitate the management of large projects. The Jack development stage is applicable to multi-agent systems composed of BDI agents, possibly interfaced with legacy systems. Note that Jack did not impose to use its own BDI model, and that other agent models can be used (but have to be implemented), while benefiting of the Jack Agent Kernel services. Both a Java programming background, and a Jack's BDI engine knowledge is required. The modularity of Jack enables a good reuse of code.

Deployment No deployment tool is provided. Deployment consists of manually launching the agent's classes.

Other At the time of writing this paper, Jack is freely available under a 60-day evaluation license. Permanent or commercial licenses are at the authors' discretion. Jack is still under evolution, and Agent Oriented Software proposes many commercial services.

Comments Jack is particular by its strong agent-programming orientation. This leads to a high versatility, agent's architecture can range from simple Java-coded reactive behaviours to full BDI, using the provided architecture or another one. Unfortunately, the documentation provided is very technical, and does not cover the methodological aspects, especially for the analysis and the design. The deployment also lacks of support. The reason for that is maybe that Jack seems to be mainly used as an internal tool by Agent Oriented Software Pty. Ltd. to provide its services.

MadKit (`http://www.madkit.org/`)
MadKit is a Java multi-agent platform built upon an organisational model. It is developed by Olivier Gutknecht and Jacques Ferber at the LIRMM (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), a public research laboratory in France. Contrarily to the other platforms presented in this paper, MadKit is mostly a MAS runtime engine, using an agent micro-kernel. The underlying organisational model is named Aalaadin [8]. A short methodology is proposed in [9].

Analysis No specific analysis method is associated with MadKit. This stage should include a functional analysis, a dependency analysis, a group context discovery and a choice of the coordination mechanisms. Any domain-specific or generic analysis method can be used.

Design Since MadKit is mainly organisation-oriented, this stage includes the definition of the organisational model (groups, roles), the interaction model (protocols, messages), and other specific entities (tasks, goals, etc). The Aalaadin model serves as a guideline for the design, but no software tools are provided. Despite the organisation orientation of the Aalaadin model, it is in fact applicable to a broad range of MAS designs. The completion of this stage is facilitated by the intuitiveness of groups and roles formations. Additionally, group and roles definitions can be reused, for example through design patterns.

Development This stage includes the choice of the agent model, its implementation, and the implementation of interaction protocol strategies. No agent model is provided, it has to be implemented in Java from scratch, or modified to work with MadKit. Since no assumption is made about the agent model, any kind of model can be used (but preferably simple ones). But as all the agent code has to be implemented in Java, it can be a heavy task to develop complex cognitive agents. Luckily, agent models implemented for MadKit can be reused across projects.

Deployment The deployment of MadKit agents takes place in the G-box, a kind of agent's sandbox, where agents can be created, modified and destroyed, with a nice graphical interface. Several G-boxes can be connected to achieve distribution across a network. A simple console mode is also available to simplify deployment. The G-box allows dynamic configuration of agent's editable features, in a similar way of JavaBeans. No global profiling is available. Agents are packaged into Jar archive, and can be mixed with other agents, allowing a high level of agent's reusability.

Other At the time of writing this paper, MadKit is free for educational use. Commercial use is at the discretion of the authors. MadKit is still under evolution, in order to add new features and to fix bugs.

Comments The main characteristic of MadKit is that it is mostly a multi-agent runtime engine. This leads to simple development and deployment, since the platform focuses on agents' infrastructure. The lack of a methodology (apart from the Aalaadin model, that is more a descriptive organisational model than a conception method) is in way of being overcomed by recent works [9], but still needs to be completed. The main default of MadKit, is that building complex agents requires writing a lot of code, since there is no pre-established agent model. On the good side, it adds flexibility, and any programmer can begin to write its own agents, even without a good multi-agent systems background, making MadKit very interesting in an educational context.

Zeus (`http://www.labs.bt.com/projects/agents/zeus/`)
Zeus is an integrated environment for the rapid building of collaborative agents applications. It is developed by the Agent Research Programme of the British Telecom Intelligent System Research laboratory. The Zeus documentation is abundant, and puts a strong emphasis on the importance of the methodological aspect of Zeus ('*The agent creation methodology is vital to the use of the Zeus toolkit*' [10]). The Zeus methodology uses the same four-stage decomposition for agent development as in our analysis, respectively domain analysis, design, realisation and runtime support.

Analysis The analysis stage, which consists of role modelling, is described in [11]. At this stage, no software tool is provided. As agents are defined by their role and by their behaviour, this stage is specifically applicable to role-oriented, rational multi-agent systems. The modelling of the roles is done with UML class diagrams and patterns, thus avoiding new formalisms and accessible to a broad audience. On the reuse aspect, a large number of commons role models are provided, covering information management, trading and business processes.

Design The agent design stage, which consists of finding solutions that fulfil the role responsibilities defined in the previous stage, is supported by three cases studies [11]: FruitMarket (Trading), PC Manufacture (Supply chain) and Maze Navigator (Rule based agents). There exists a software tool at this stage. The underlying Zeus agent model limits the design to task-oriented, goal-driven, collaborative agents. Since the process of finding a solution to a given problem is the most difficult thing to systematise, the design stage mainly requires design skills, but is not technically difficult. The three case studies are a good starting point for reuse. The simplicity of the design formalism used by Zeus facilitates design reuse, but lacks of formalism (mainly composed of natural language statements).

Development The agent development stage is both covered by the three supplied case studies and the Application Realisation Guide [11]. The five activities involved for developing agents are both supported by graphical software tools These five activities are: Ontology Creation, Agent Creation, Utility Agent Creation, Task Agent Configuration and Agent Implementation. These activities require a good knowledge of the tools, which are hopefully well documented. Of course this stage is only

applicable to the Zeus agent model. Ontologies can be easily reused. There is still no support for agent reuse across projects. The general modularity of the development tool enables the reuse of frequently used agent functionalities, but adding a new functionality requires to return to the (Java) code (for example, to add a new agent co-ordination strategy).

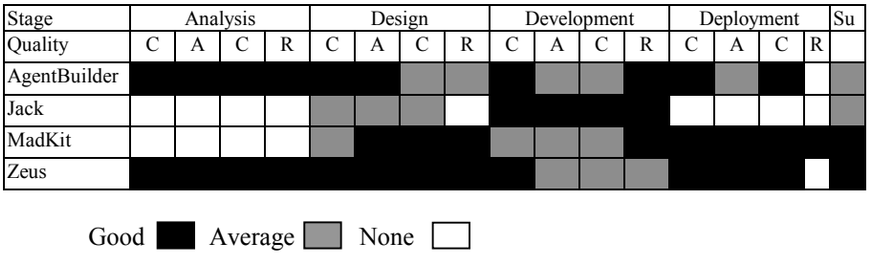
Deployment The deployment stage is documented in the Runtime Guide [11]. This document describes how to launch the generated MAS, and how to use the Visualiser tool. There are also some considerations about the art of debugging. The visualiser tools visualises the MAS from different points of view: organisation and interactions in the society, global task decomposition, statistics and internal states of agents. It is also possible to control any individual agent states, and even to configure agents at run-time. The deployment tools use user-friendly graphic interfaces that facilitate the deployment. Modifications to the multi-agent systems' society composition or localisation require recompilation, thus making the deployment reuse next to impossible.

Other The complete Zeus toolkit is available for download on the Zeus web site. Zeus is Open Source software, hence is free of charge for academic and industrial users. No official support is provided. A Zeus mailing list is available.

Comments The main particularity of Zeus is its complete integration of all the stages from design to deployment. It provides theoretical and practical tools, uses actual techniques of software engineering (design patterns, UML), and its methodological documents focus on the *how to* and not only on the *what to do*. However, even if Zeus is modular, there is only one agent model supported, which limits the range of possible designs of multi-agent systems. Also, as any complex tool, Zeus is long and difficult to master, but with the benefice of a great productivity once it is mastered.

5 Discussion

As we have seen with the four platforms presented here, the growing interest in multi-agent development platforms has lead to very interesting tools. Of course this is only a comparison between some specific platforms, and since we believe there is no absolute MAS methodology and platform, any comparison has some bias. The following table illustrates the qualitative values of the several criteria (Completeness, Applicability, Complexity, Reusability) we have used to analyse the four platforms (AgentBuilder, Jack, MadKit, Zeus).



From the methodological point of view, we observe that there exist big differences between platforms, from the Zeus's *vital* methodology, to the quasi-absence of methodological material or tools. Often, the starting point of a platform is an *implementation* tool, but such a tool is not self-sufficient. The programmer needs a manual to use it. A descriptive manual is not enough, because the fundamental question a programmer asks himself is "How do I build a multi-agent system from that specification?" and this question is not answered by a simple description of the tool. The methodological documentation of a platform must help the programmer, by defining simple tasks that, step by step, lead from the analysis to the deployment of his application.

From the technical point of view, different solutions have been chosen: AgentBuilder and Zeus are more like "BDI editors", whilst Jack is an agent programming language and MadKit a multi-agent runtime infrastructure. Each kind of solution has its advantages and drawbacks. BDI editors are very efficient because they allow programming in a high level of abstraction, but are committed to single agent architecture. On the other hand, agent programming languages are more versatile, but at the expense of more code writing, because the level of abstraction is lower. A better solution would be a modular platform, that would give to the programmer a rich library of agent models, where each model would have its own dedicated editor. This would combine good versatility and high-level programming, even at the price of a higher platform development cost.

We have also noted all these platforms lacks some important aspects of multi-agent systems, such as the notion of environment, which can play a very important role in some applications [12], and have difficulties to apprehend the agent society as a whole. Coordination and cooperation aspects are at best limited to the definition of groups and the design of interaction protocols. The engineering of societies, especially emergent organisations [13], continue to show too much conceptual, methodological and technical difficulties to become a standard of multi-agent platforms.

We are currently working in the Magma group on modular multi-agent platforms that address as much as possible the proposed criteria : the MASK [14] platform and its successor, Volcano, currently under development. In these platforms, multi-agent systems are composed of four parts: Agent, Environment, Interaction and Organisation, corresponding to the Vowels methodology [15]. Each part is represented in the platform as a toolbox, containing a library of models from which the developer can choose the most adequate one and then instanciate it. We are also developing the accompanying methodology, to cover as much as possible the four

stages. This should result into a revised platform in some future, which will better reflect our Vowels methodology [15].

References

1. Federico Bergenti and Agostino Poggi. Exploiting UML in the Design of Multi-Agent Systems. In this volume.
2. Y. Shoham. AGENT-0: A simple agent language and its interpreter. *In Proceedings of the Ninth National Conference on Artificial Intelligence*, Vol II (pp. 704-709), Anaheim, CA, MIT Press, 1991.
3. Y. Shoham. Agent Oriented Programming. Artificial Intelligence, 60(1), pp. 51-92, North-Holland, 1993.
4. S. R. Thomas. PLACA, an Agent Oriented Programming Language. Ph.D. Thesis, Stanford University, 1993.
5. Reticular Systems. AgentBuilder User's guide. External documentation, `http://www.agentbuilder.com/`, August 1999.
6. Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal Specification of dMARS. In Singh et al, editors, *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL'97)*, LNAI, Vol. 1365, pp. 155-176, Springer, 1998.
7. P. Busetta, R. Rönnquist, A. Hodgson, A. Lucas. JACK Intelligent Agents – Components for Intelligent Agents in Java. Updated from AgentLink Newsletter #2, `http://www.agent-software.com.au/`, October 1999.
8. J. Ferber and O. Gutknecht. A meta-model for analysis and design of multi-agent systems. *Proceedings of the 3rd International Conference on Multi-Agent Systems*, (ICMAS'98), IEEE, pp. 155-176, August 1998.
9. O. Gutknecht and J. Ferber. Vers une méthodologie organisationnelle pour les systèmes multi-agents. In *Actes des JFIADSMA'99 (Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents)*, Saint-Denis, Reunion, 1999
10. J. Collis, D. Ndumu. The ZEUS Technical Manual. external documentation, `http://www.labs.bt.com/projects/agents/zeus/`, September 1999.
11. J. Collis, D. Ndumu, and S. Thompson. ZEUS Methodology Documentation, Role Modelling Guide, Three case studies, Application Realisation Guide, Runtime Guide. `http://www.labs.bt.com/projects/agents/zeus/`, August-November 1999.
12. H. Van Dyke Parunak, Sven Brueckner, John Sauter and Robert S. Matthews. Distinguishing Environmental and Agent Dynamics: A Case Study in Abstraction and Alternate Modeling Technologies. In this volume.
13. Cristiano Castelfranchi. Engineering Social Order. In this volume.

14. M. Occello, C. Baeijs, Y. Demazeau, and J. L. Koning. MASK : An AEIO Toolbox to Develop Multi-Agent Systems. In Cuena et al editors, *Knowledge Engineering and Agent Technology, IOS Series on Frontiers in Artificial Intelligence and Applications*, 2000.
15. Y. Demazeau, M. Occello, C. Baeijs, and P.-M. Ricordel. Systems Development as Societies of Agents. In Cuena et al editors, *Knowledge Engineering and Agent Technology, IOS Series on Frontiers in Artificial Intelligence and Applications*, 2000.

Exploiting UML in the Design of Multi-agent Systems

Federico Bergenti and Agostino Poggi

Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma
Parco Area delle Scienze 181A, Parma, Italy
{Bergenti,Poggi}@CE.UniPR.IT

Abstract. A basic concept of software engineering is that a system can be described at different levels of abstraction. Agent-oriented software engineering introduces a new level of abstraction, called the agent level, to allow software architects modelling a system in terms of interacting agents. This level of abstraction is not yet supported by an accepted diagrammatic notation even if a number of proposals are available. This work shows how UML can be exploited to model a multi-agent system at the agent level. In particular, it presents a set of agent-oriented diagrams intended to provide an UML-based notation to model: the architecture of the multi-agent system, the ontology followed by agents and the interaction protocols used to co-ordinate agents. The presented notation exploits stereotypes to associate an agent-oriented semantic with class and collaboration diagrams. The benefit of using stereotypes rather than extending UML to provide an agent-oriented semantic is that the presented notation can be used with any off-the-shelf CASE tool.

1 Introduction

The research on agent-oriented software engineering starts from the possibility to model a software system at the agent level of abstraction [8]. This level of abstraction considers agents as atomic entities that communicate to implement the functionality of the system. This communication is supported by an agent communication language, such as FIPA ACL [5] or KQML [2], and by an ontology used to associate a meaning with content messages. Even if the agent level is the natural level of abstraction for describing a multi-agent system, the lack of accepted diagrammatic notations and design tools might prevent the multi-agent architect from exploiting its benefits. The research community of agent-oriented software engineering is currently addressing this problem investigating the possibility of extending UML to support the basic agent-oriented concepts such as agent, ontology and interaction protocol. This research trend has led to various extensions of UML [3, 12, 13, 15]. Anyway, such notations are not yet completely accepted because they are not supported by CASE tools. It is worth noting that one of the main goals of the AUML initiative [13] is to encourage CASE-tools vendors to support an agent-oriented notation in their products.

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 106-113, 2000.

This work tackles the problem of defining a diagrammatic notation to support the design at the agent level exploiting an UML-based notation that can be managed by any off-the-shelf CASE tool. In particular, four agent-oriented diagrams are introduced taking advantage of stereotypes, the customisation means offered by UML. This allows providing an agent-oriented semantic to UML as a straightforward generalisation of its well-known, and accepted, object-oriented semantic, and therefore the software architect should feel comfortable with it. The presented diagrams are meant to support the basic agent-oriented abstractions, but they cannot be considered sufficient to support a complete agent-oriented model of the software life cycle. Various initiatives [11, 16] are intended to provide a methodology to support the agent-oriented development process and the ideas presented in this work could be used to support such methodologies with a UML-based notation.

The rest of this work is organised as follows: section 2 introduces *ontology diagram* as a means to provide the multi-agent architect with a means to model ontologies. Section 3 tackles the problem of modelling the architecture of a multi-agent system introducing *architecture diagrams*. Section 4 completes the presented set of diagrams with *role diagrams* and *protocol diagrams*. These diagrams can be used jointly to model the interaction protocols that the multi-agent architect may use to coordinate agents. In order to show the possibilities and the features offered by the proposed notation, the agent-based service defined in the FIPA Audio/Video Entertainment and Broadcasting (*FIPA AVEB*) specification [4] is taken into account. This specification is sufficiently complex to represent a valid test case for the proposed notation and the comparison with the textual description found in FIPA documents should clarify the benefits of the presented approach.

2 Ontology Diagrams

The design of a multi-agent system at the agent level of abstraction requires defining a model of the world in which agents live. Agents exploit this model to reason about the world and to talk about it. The agent level of abstraction does not deal with the internal architecture of agents, as they are considered atomic, and therefore the required model of the world is intended only to support the communication between agents.

The problem of describing the world to agents is traditionally solved providing them with an ontology that outlines a model of the world in terms of entities and relations between such entities [1, 6, 7, 14]. UML can be used to model ontologies exploiting ontology diagrams, i.e., class diagrams whose elements are associated with an ontology-oriented semantics. Such semantics are straightforward as they resemble the ones employed in conceptual diagrams [10]. An ontology diagram allows describing the entities belonging to an ontology in terms of classes. Such *entity classes* represent classes of entities comprised in the ontology, just like ordinary classes in a class diagram represent classes of objects. Moreover, ontology diagrams allows the multi-agent architect defining the relations between the entities belonging to an ontology exploiting relations between the corresponding entity classes. Such relations are mapped to UML public relations between entity classes.

The agent level of abstraction does not deal with implementation details as they are normally tackled at the object level. This implies that entity classes are not allowed containing private and protected methods and attributes. Similarly, class diagrams allow associating a set of public methods to a class of objects to specify the messages that objects may exchange to implement the functionality of the system. At the agent level of abstraction this is not allowed because the agents are the actors that communicate to implement the system. This approach restricts the world to comprise only passive entities that cannot communicate directly with agents. Agents might be triggered by such entities only because they are capable to sense them.

Entity classes defined in an ontology diagram are characterised only in terms of public attributes. Such attributes are used to model the structure of the entities comprised in the ontology, just like they are used in conceptual diagrams. Figure 1 shows an ontology diagram describing a subset of the ontology defined in the FIPA AVEB specification. This diagram defines a class of entities, called *FIPAAVDescription*, characterised by a string called *title*. The stereotype *entity* is introduced to allow the multi-agent architect keeping the entities belonging to the ontology apart from the objects used at the object level. Moreover, this stereotype allows an agent-enabled CASE tool to employ a different diagrammatic convention for object classes and entity classes, just like they normally do for actors and objects.

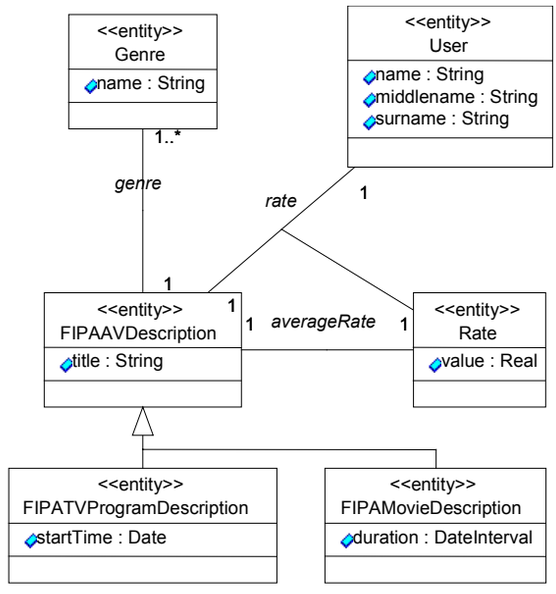


Figure 1. Part of the ontology diagram for the FIPA AVEB ontology

Ontology-description languages allow modeling the relations between the entities comprised in an ontology. Ontology diagrams support this feature exploiting public relations between entity classes. For example, figure 1 shows that the classes *FIPATVProgramDescription* and *FIPAMovieDescription* extend the class *FIPAAVDescrip-*

tion. This means that such classes contain all attributes that characterize entities belonging to the class *FIPAAVDescription*.

The main purpose of ontology diagrams is to model the ontology followed by agents at the agent level of abstraction and the multi-agent architect may take this into account to produce useful models. As noted above, ontology diagrams should contain the features needed to support the communication between agents. This can be accomplished treating relations between the entity classes as predicates defined over such classes. This approach allows the multi-agent architect exploiting an ontology diagram as a description of some of the elements, i.e., the entity and the predicates, that agents may use to create content messages. For example, figure 1 shows a relation called *averageRate* between an entity belonging to the class *User* and an entity belonging to the class *FIPAAVDescription*. This relation is chosen because the FIPA AVEB specification requires agents understanding messages like:

```
(inform
   :sender      guide-agent
   :receiver    control-agent
   :content
      (averageRate (Rate :value 0.8)
                   (FIPAAVDescription :title "Batman")))
```

3 Architecture Diagrams

The research community of agent-oriented software engineering has not yet identified an accepted process to support the design of a multi-agent system, even if various proposals are available [11, 16]. All proposed processes emphasise the importance of modelling a multi-agent system in terms of an architecture that can be instantiated to create a configuration. Such architecture is composed of a set of *agent classes*, or agent roles [9], and each class represents a class of agents with a precise set of responsibilities.

UML can be used to model the configuration of a multi-agent system exploiting deployment diagrams. Besides, the modelling of its underlying architecture is not that easy. In fact, an architecture is generally specified by means of a textual description. Such description contains a list of agent classes and a set of relations connecting them. Moreover, each class is associated with a set of responsibilities and with a set of messages that an agent belonging to that class is requested to understand in the scope of an agent communication language. We propose to use architecture diagrams to model the architecture of a multi-agent system. These diagrams are class diagrams that contain agent classes and relations between such classes. The complete model of an agent class requires describing the set of features used to characterise an agent belonging to that class. Such features include:

1. the supported interaction protocols;
2. the accepted content messages, taking into account the ontology;
3. the semantic of such message.

Architecture diagrams support the latter two features leaving the description of interaction protocols to protocol diagrams, as described in the following section, or to AUML protocol diagrams [13].

An agent class can contain only public methods corresponding to the actions that an agent belonging to that class can be requested to perform. Such methods must be declared without any return values, i.e., they must be declared void. Moreover, the parameters passed to these methods must belong to entity classes defined in an ontology diagram. The list of actions associated with agent class does not include the performatives of the agent communication language because these are almost embedded in chosen the agent model. As an example, figure 2 shows the complete architecture diagram of the system defined in the FIPA AVEB specification. This diagram contains seven classes and just for one of them the FIPA AVEB specification requires an action. Only agents belonging to the class *TunerWrapper* are requested to support an action, called *invoke*, to wrap concrete commands directed to a TV tuner. This action is understood only when supplied along with two parameters: a tuner, i.e., an entity belonging to the class *Tuner*, and a command expressed as a *String*. Therefore, this diagram allows an agent belonging to the class *TunerWrapper* to accept messages like:

```
(request
   :sender     control-agent
   :receiver   tuner-wrapper-agent
   :content
     (action invoke (tuner   :name tuner1)
                    (command "start")))
```

The actions that the multi-agent architect associates with the agent classes complete the elements provided by ontology diagrams to create valid content messages. An agent can understand all messages composed using the entities and predicates found in ontology diagrams and can be requested to perform the actions specified in its class. This approach restricts agents understanding only messages based on predicate logic. While this is a limitation of the considered logic framework, it does not limit strongly the set of messages that agents may wish to exchange in a real-world systems. Therefore, the development of a more comprehensive logic framework is considered for a future work.

Architecture diagrams contain also relations between agent classes to allow the multi-agent architect modelling the relations between agents. These relations are mainly used to express simple acquaintance as it is common to promote flexibility in a multi-agent system avoiding the use of relations to spread the responsibility across agents. For example, figure 2 shows that the FIPA AVEB specification requires the control agent knowing all other agents in the multi-agent system, while the VoD controller agent knows only the control agent.

4 Protocol Diagrams and Role Diagrams

Interaction protocols are considered a valuable tool the multi-agent architect may exploit in the design of a system at the agent level. They allow modelling the interac-

tions between agents without taking into account the formal semantic of the chosen agent communication language. Moreover, they represent off-the-shelf solutions to a large number of problems.

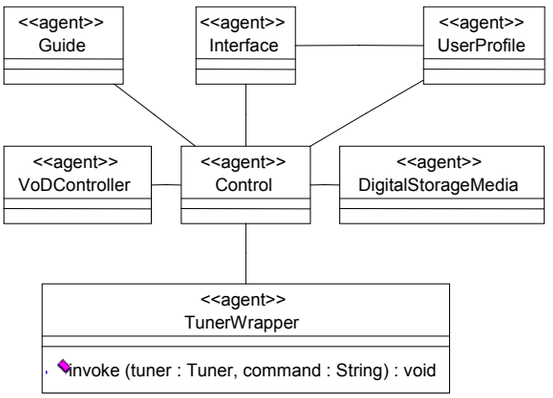


Figure 2. Architecture diagram of the system defined in the FIPA AVEB specification

Object protocols, the object-oriented counterpart of interaction protocols, are not considered a fundamental element in the design of a system at the object level. This is the main reason why UML does not provide diagrams to model object protocols. This has led to the assumption that UML should be extended to model interaction protocols at the agent level of abstraction. For example, AUML [13] defines a diagrammatic notation to modelling interaction protocols based, but not compatible, with sequence diagrams. A sequence diagram expresses a sequence of messages exchanged by a set of objects, while an AUML diagram models all sequences of messages that a set of agents may exchange in the scope of an interaction protocol.

We propose to use protocol diagrams and role diagrams as an UML-based notation to model interaction protocols. Such diagrams are based on collaboration diagrams and class diagrams respectively. A protocol diagram is a collaboration diagram that comprises only classes declared in a role diagram. These classes are labelled with the stereotype *role* to keep them separated from other kinds of classes, such as agent or entity classes. A role class models a role played by an agent in an interaction protocol. This role is meaningful only in the scope of the protocol in which it is defined and it is described in terms of the communicative acts that an agent playing that role must support. As an example, figure 3 shows the FIPA reactive contract net protocol defined in the FIPA AVEB specification.

5 Conclusions

Software engineering is founded on the possibility to model a system at different levels of abstraction. Agent-oriented software engineering introduces a new level, called the agent level, to allow the software architect describing a system in terms of inter-

acting agents. At this level of abstraction, an agent is considered as an atomic entity that communicates with all other agents comprised in the multi-agent system exploiting an agent communication language. Besides, the multi-agent architect may not exploit the benefits offered by this new level of abstraction for the lack of accepted notations and tools. This work shows an UML-based notation that can be used to model a system at the agent level. The choice of exploiting UML in our notation is made to allow software architects understanding and possibly adopting this work easily. Moreover, the concepts introduced in this work are agent-oriented generalisations of well-known, and accepted, object-oriented concepts and therefore the choice of UML seems forced.

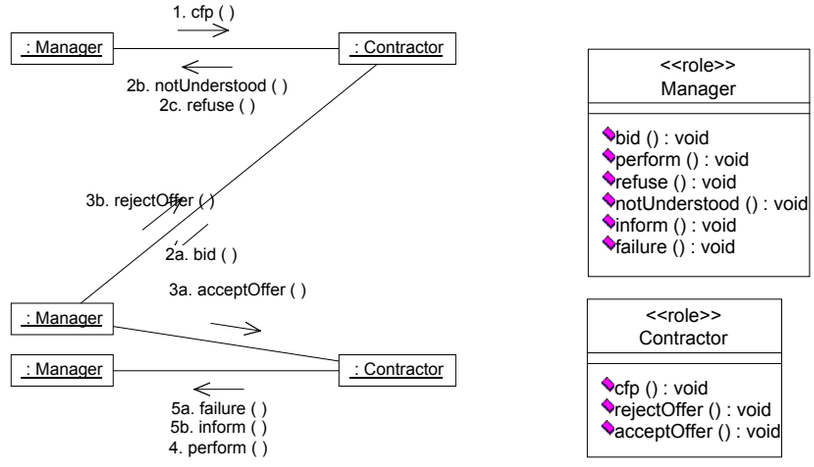


Figure 3. Protocol diagram and role diagram for the FIPA reactive contract net protocol

The presented notation relies on the possibility to provide an agent-oriented semantic to the elements comprised in class and collaboration diagrams. In particular, four agent-oriented diagrams are introduced exploiting the customisation means provided by UML to support domain-specific diagrams. Such diagrams can be used to model the basic elements characterising the agent level of abstraction: the architecture of the multi-agent system, the interaction protocols supported by agents and the ontology followed by agents. Architecture diagrams allows modelling the architecture of the multi-agent system in terms of a set of agent classes connected through relations. Each class is characterised by the actions that an agent belonging to it can be requested to perform. The relations between classes may be used to express the network of acquaintance that an agent can gain in the architecture of the multi-agent system. Role diagrams and protocol diagrams are intended to support the multi-agent architect in the definition of interaction protocols. Role diagrams can be used to specify the roles that agents may play in a protocol, while protocol diagrams model an interaction protocol defined over a set of roles as the set of all possible conversations compatible with it. Ontology diagrams allows defining a model of the world composed of entities and relations. These relations can be used to specify the predicates that agents may use

to communicate and therefore ontology diagrams can be used to model the content messages that agents are allowed to use to communicate.

References

1. S. Cranefield and M. Pruvis, *"UML as an Ontology Modelling Language"*, in Proceedings of the Workshop on Intelligent Information Integration, 1999.
2. T. Finin, Y. Labrou and J. Mayfield, *"KQML as an Agent Communication Language"*, in J. M. Bradshaw (Ed.) Software Agents, MIT Press, 1997.
3. FIPA Technical Committee C, *"Extending UML for the Specification of Agent Interaction Protocols"*, response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, 1999.
4. FIPA, *"FIPA '97 Specification Part 6: Audio/Video Entertainment and Broadcasting"*, available at http://www.fipa.org.
5. FIPA, *"FIPA '99 Specification Part 2: Agent Communication Language"*, available at http://www.fipa.org.
6. M. R. Genesereth and R. E. Fikes, *"Knowledge Interchange Format - Version 3 - Reference Manual"*, technical report Logic-92-1, Stanford University, 1992.
7. M. R. Genesereth, N. Singh and M. Syed, *"A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation"*, International Journal of Cooperative Information Systems 4(4):339-367, 1995.
8. C. A. Iglesias, M. Garijo and J. C. A. González, *"Survey of Agent-Oriented Methodologies"*, in Proceedings of the Workshop on Agent Theories, Architectures and Languages, 1998.
9. E. A. Kendall, *"Role Model Designs and Implementations with Aspect Oriented Programming"*, in Proceedings of the 1999 Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM Press, 1999.
10. C. Larman, *"Appling UML and Patterns"*, Prentice Hall, 1997.
11. MESSAGE Consortium, *"Deliverable 1: Initial Methodology"*, deliverable of the EURESCOM Project P907-GI, 2000.
12. J. Odell and C. Bock, *"Suggested UML Extensions for Agents"*, response to the OMG Analysis and Design Task Force UML RTF 2.0 Request for Information, 1999.
13. J. Odell, H. Van Dyke Parunak and B. Bauer, *"Representing Agent Interaction Protocols in UML"*, in Proceedings of Agents 2000, 2000.
14. P. F. Patel-Schneider and B. Swartout, *"Description-Logic Knowledge Representation System Specification"*, DARPA KSE technical report, 1993.
15. J. Treur, *"Methodologies and Software Engineering for Agent Systems"*, available at http://www.cs.vu.nl/~treur.
16. M. Wooldridge, N. R. Jennings and D. Kinny, *"The Gaia Methodology for Agent-Oriented Analysis and Design"*, in Journal of Autonomous Agents and Multi-Agent Systems 3 (3) 285-312, 2000.

Formal Specification and Prototyping of Multi-agent Systems

Vincent Hilaire[1], Abder Koukam[1], Pablo Gruer[1], and Jean-Pierre Müller[2]

[1] Département Génie Informatique-Systèmes Et Transports
Université de Technologie de Belfort Montbéliard
4 rue du château, 90010 Belfort Cedex, France
{vincent.hilaire,abder.koukam,pablo.gruer}@utbm.fr
[2] Institut d'Informatique et d'Intelligence Artificielle, Université de Neuchâtel
Emile-Argand 11, CH-2007 Neuchâtel, Switzerland
jean-pierre.muller@info.unine.ch

Abstract. This paper presents a multi agent-oriented prototyping approach. It is a generic approach, applicable to a wide range of multi-agent systems. This approach relies on a few assumptions, the most important is that MAS must be described by an organizational model which semantics is given in term of a formal framework. This model allows for a simple description of both individual and collective multi-agent system aspects. The framework we use to give a formal description of this model is based on a multi-formalism approach. We illustrate this approach through a case study.

Keywords: Agent, Specification, Prototyping

1 Introduction

Agent-based systems are new paradigm for modeling and building many computer systems ranging from complex distributed systems to intelligent software applications. It proposes new ways for analyzing, designing and implementing such systems based upon the central notions of agents, their interactions and the environment which they perceive and in which they act. Although many Multi-Agent Systems (MAS for short) have been designed, there is a crucial lack concerning specification and development methodologies.

Process of specification is fundamental to handle the complexity related to build such systems and specifying the desirable behavior of MAS before their implementation phase. The specification process must fulfill two roles. The first is to provide the underlying rationale for the system under development. The second is to guide subsequent design, implementation and verification phases. A variety of specification formalisms are available in the multi-agent field. Such formalisms put the emphasis on the first role and do not provide a basis to fulfill the second. As stated in [4], they are often abstract and unrelated to concrete computational models. We believe that one way to bridge the gap between the

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 114–127, 2000.

abstract and the concrete level is to build the system specification using a prototyping process [15]. This process provides a support for incremental specification leading to an executable model of the system being built. Indeed, in many areas of software and knowledge engineering, the development process putting emphasis on prototyping and simulation of complex systems before their effective implementation is proven to be a valuable approach.

The purpose of this paper is to present a formal approach to MAS that fits in with prototyping and simulation oriented processes. The first step towards such approach is to choose or to build a formalism for specifying MAS. Harel and Pnueli [12] split systems in two classes: transformational systems which can be described by a functional mapping input to output values and reactive systems which do not compute functions but perform a continuous interaction with their environment. Due to their complexity, MAS have reactive and transformational features. A formalism which: specifies easily and naturally both aspects, enables prototyping and simulation and guides the implementation phase which is to be defined yet. We have thus chosen to use a multi-formalism approach that results of the composition of Object-Z [5] and statecharts [13]. This formalism enables the specification of reactive and transformational aspects of MAS and their prototyping by simulation.

Even though it has enough expressive power to specify MAS aspects, this language does not provide any methodological guidelines. A specification method is essential to manage MAS complexity by decomposition and abstraction. We use an organizational model [7,20] that consider organizations, interactions and roles as first class citizen. This model allows to go from the requirements to detailed design and helps to decompose a MAS in terms of roles and organizations. Model concepts are specified by a framework which has to be refined to specify a MAS particular application. These concepts are illustrated by the Aphtous Ulcer Fever example which has been presented in [6]. This example consists in cattles disease modeling by using roles and organizations.

The paper is organized as follows. Section 2 introduces the methodological approach proposed which results from an organizational model and formal specification framework. Section 3 gives the specification of the Aphtous Ulcer Fever example. Section 4 is devoted to a summary, and prospects for further research.

2 Specification Approach

2.1 Role/Organization Model

Our specification approach uses an organizational model which is based on three interrelated concepts: role, interaction and organization. Roles are generic behaviors. These behaviors can interact mutually according to interaction pattern. Such a pattern which groups generic behaviors and their interactions constitutes an organization. Organizations are thus descriptions of coordination structures. Coordination occurs between roles as and when interactions takes place.

In this context, an agent is only specified as an active communicative entity which plays roles [7]. In fact agents instantiate an organization (roles and in-

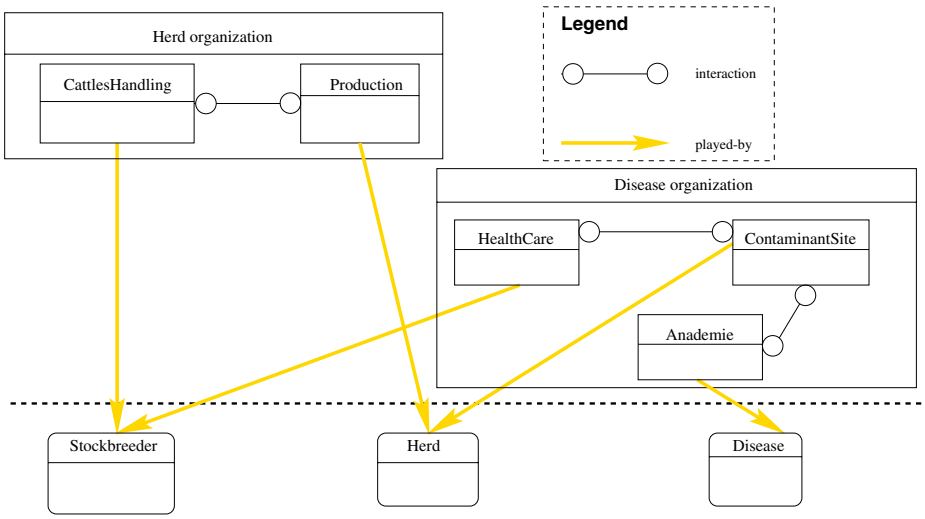


Fig. 1. Ulcer Aphtous Fever semi-formal model

teractions) when they exhibit behaviors defined by the organization's roles and when they interact following the organization interactions.

An agent may instantiate one or more roles and a role may be instantiated by one or more agents. The role playing relationship between roles and agents is dynamic. We think this model is a basis for the engineering of societies of agents and what Castelfranchi calls "social order" [3].

We make no assumptions on agent architectures. The generality of the agent definition allows the specification of many agent types. More specific choices can be introduced in more accurate models. This model enables a modular approach by prototyping separate parts of the MAS. The behavior of the MAS as a whole is the result of the role playing by agents.

This methodology deals with both the macro (organizations) level and micro (roles) level. Moreover, this MAS decomposition enables the prototyping of only part of the MAS. In fact, the smallest entity one can prototype is a role.

The roles, interactions and organizations of the Aphtous Ulcer Fever example are described by figures 1. To describe these organizations we borrow from the OOram notation [18].

A box that represents an organization encloses a set of boxes representing its roles. An interaction is materialized by a line connecting two roles. On each extreme of this line there may be nothing, a circle or a double circle to indicate the interactions arities. In our example, arities are all one to one, i.e concrete interactions always happen between two agents playing the respective roles. We present only a part of the Aphtous Ulcer Fever MAS model of [6].

In the part we present, there are two organizations: *Herd* organization which deals with aspects related to cattles and *Disease* organization which models aspects related to disease spread and cure.

The former organization is composed of two roles: *CattlesHandling* (placing cattles either in farm or field and selling calfs) and *Production* (simulating calfs birth). The latter organization is composed of three roles: *HealthCare* (searching and cure of infected animals), *ContaminantSite* (simulating of possible contagions) and *Anademie* (simulating self-cures and contaminations).

Three agents type play these roles. *STOCKBREEDER* agent plays *CattlesHandling* and *HealthCare* roles. *HERD* agent plays *Production* and *ContaminantSite* roles. Eventually *DISEASE* agent plays *Anademie* role. The roles played by these agents are shown in figure 1 by the arrows from roles to agents.

While this model is represented in a well structured and easy to understand manner, like the one of [1] it lacks of formal semantics and means of rigorous analysis such as verification and prototyping. The section 2.3 introduces a formal framework fulfilling those needs. This formal framework is based upon a formalism presented in the next section.

2.2 Multi-formalism Based Specification Language

Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to modeling all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend one formalism.

The multi-formalism approaches [21,17] compose two or more formalisms in order to specify more easily and naturally than with a single formalism. Indeed, the multi-formalism approach deals with complexity by applying formalisms to problem aspects for which they are best suited and to prove properties with proofs rules and transformation techniques available in a specific formalism.

Our choice is to use Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects.

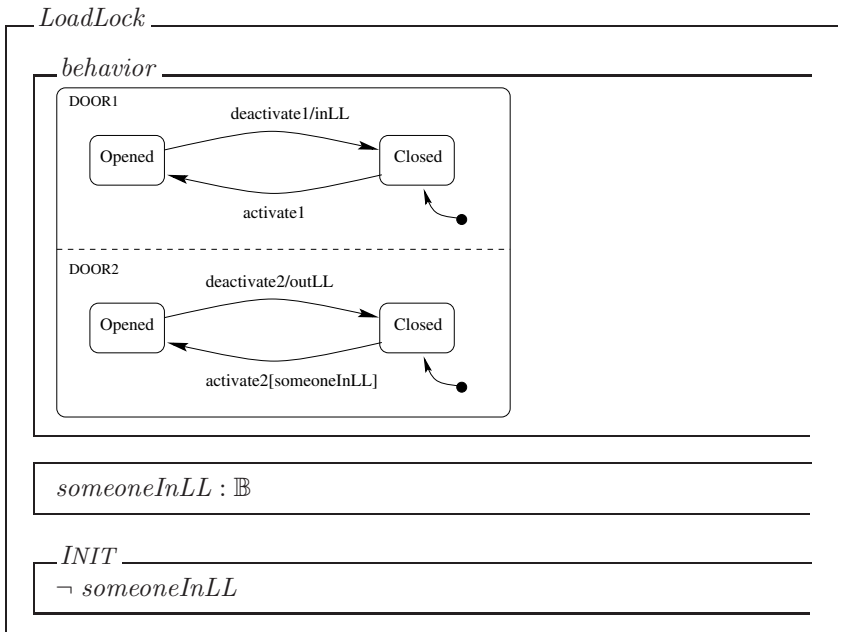
Object-Z extends Z with object-oriented specification support. The basic construct is the class which encapsulates state schema with all the operation schemas which may affect its variables.

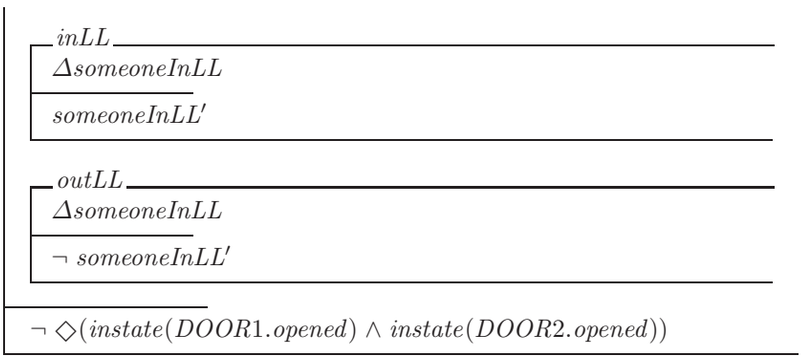
Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both language have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.

Our method for composition of Object-Z and statecharts relies on the meta-method of Paige [17]. The main step of this method is the definition of an heterogeneous basis which is a set of notations, translations and formalizations that provides a formal semantics to multi-formalism specifications. In our case the main concept of the heterogeneous basis is the integration of statecharts in Object-Z classes. We have extended the expressive capabilities of each formalism by features available in the other. The role of the heterogeneous basis is to provide formal means of expression without translating a formalism in the other. In other words the heterogeneous basis furnishes mean of communication between partial specifications written in either Object-Z or statecharts.

The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes the possible states of the object and events which may change these states. A statechart included in an Object-Z class can use attributes and operations of the class. The sharing mechanism used is based on name identity. Moreover, we introduce basic types $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes. These types also belong to the heterogeneous basis.

The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between $DOOR1$ and $DOOR2$. The first door reacts to *activate*1 and *deactivate*1 events. When someone enter the *LoadLock* he first activate the first door enter the *LoadLock* and deactivate the first door. The transition triggered by *deactivate*1 event execute the *inLL* operation which sets the *someoneInLL* boolean to true. Someone which is between the first and the second door can activate the second door so as to open it. The temporal invariant at the end of the class specifies that the statechart must not be in $DOOR1.opened$ and $DOOR2.opened$ states simultaneously. This invariant uses the predicate $instate(S)$ which is true whenever S state is active.





The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.

The result of the composition of Object-Z and statecharts seems particularly suited in order to specify MAS. Indeed, each formalism have constructs which enable complex structure specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with. In fact, available constructs enable natural specification of "low" level aspects inherent to MAS. Higher level aspects like coordination are expressed by roles, interactions and organizations classes which we present in the following section.

2.3 Framework

For specifying MAS described with the role/organization meta-model we build a framework based upon Object-Z and statecharts. Our framework is composed of a set of such classes which specify all meta-model concepts: role, interaction and organization.

The structure of the framework as a class hierarchy allows to inherit from defined classes in order to specify more specific model concepts or to extend existing concepts. Every model concept is given a formal semantics by a class of the framework. The framework provides specifiers with a structured approach committed to the organizational model.

We present three classes of the framework: *Role*, *Interaction* and *Organization*. The first concept of the model is role. A role is defined by a behavior schema which has to be refined in order to define the role behavior. Typically it's in a specialized behavior schema where the specifier integrate a statechart. State schema is composed by sets of *Attribute*, *Event* it can react to and *Action* it can execute.

Role

behavior

$attributes : \mathbb{P}\, Attribute$
$stimulus : \mathbb{P}\, Event$
$actions : \mathbb{P}\, Action$

An interaction is composed of two roles and an interaction content, namely *content*, from a role origin of the interaction to a role destination of the interaction. Moreover, one may extend interaction to take into account more than two roles or more complex interactions involving plan exchange.

An organization is composed of a set of roles and their interactions. All interactions must only take places between two roles belonging to the same organization.

Interaction

$orig == first\ roles$
$dest == second\ roles$

$roles : Role \times Role$
$content : Event$
$\forall\, e \in msg \bullet$
$(e \in (dest.stimulus)$
$\wedge (e \in orig.actions))$

Organization

behavior

$roles : \mathbb{P}\, Role$
$env : \mathbb{P}\, envRole$
$interactions : \mathbb{P}\, Interaction$

$\forall\, i \in interactions \bullet (i.orig \in roles \wedge i.dest \in roles)$

In order to specify interaction protocols within an organization one has to refine the *Organization* class by introducing, for instance, temporal logic predicates known as temporal invariants.

3 Specification and Prototyping

3.1 Aphtous Ulcer Fever Specification

This section gives a formal description about Aphtous Ulcer Fever specification by refinement of the specification framework described previously. This formal description is then used as a basis for prototyping.

Production class inherits from Role. As described in the following it replace behavior sub-schema by PRODUCTION statechart. PRODUCTION behavior consists in waiting in *Idle* state for january to occur and then the *Idle* to *Birth* transition is triggered. This transition execute the *birth* operation which simulate calfs birth. On February the *Birth* to *Idle* transition is triggered. The class state is composed of sets of *Animal*. Each set specify an animal type, cattles for example are all animals, this constraint is specified by the expression $cattles = adult \cup calfs$. These sets belong to the attributes set of the role. The birth operation modifies calfs set. So calfs belongs to the Δ-list of the operation. This operation augments the calfs set size in order to simulate the increase of calfs count.

The same remarks stand for the role CattlesHandling except behavior subschema is replaced by CATTLESHANDLING statechart and *sellCalfs* operation diminishes calfs size. CATTLESHANDLING statechart specify two parallels process (shown as dashed line): cattles localization and calfs selling. Indeed, calfs can be either in farm or in field and every october calfs are to be sold. The entering of *Farm* and *Field* states trigger respectively $contagion - possible$ and $contagion - impossible$ events.

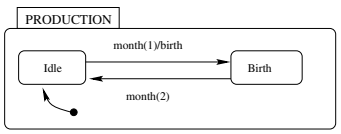
HealthCare role consists, each month, in simulating herd scanning. Indeed, the *Idle* to *Testing* transition is triggered every month and if infected animals are detected, a cure is initiated by executing the *Cure* operation. The cure is efficient as soon as it is done and will last for three month. If there are no infected animals the *Testing* to *Idle* transition is triggered.
Anademie role describes the disease evolution which consists, for any month and if infected animals are detected, of two parallel processes: contamination and cure. The former tests if contagion is possible. If contamination is possible then the *noContamination* to *Contamination* transition is triggered and then, each month, the infected animals number may augment. Whenever there are infected animals the latter process simulates their cure, in other word it may decrease the infected animals number. If there are no infected animals the role is in *Default* state.

Eventually, *ContaminantSite* role decrement the reminiscence of cure and establish if contagion is possible.

Production

$Role[PRODUCTION/behavior]$

PRODUCTION



$cattles : \mathbb{P}\, Animal$
$adult : \mathbb{P}\, Adult$
$calfs : \mathbb{P}\, Calf$

$\{cattles, adult, calfs\} \in attributes$
$cattles = adult \cup calfs$

birth

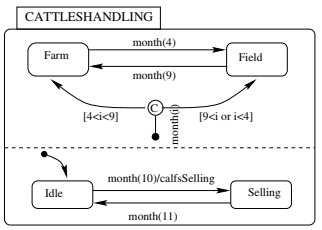
$\Delta calfs$

$\#calfs' \geq \#calfs$

CattlesHandling

$Role[CATTLESHANDLING/behavior]$

CATTLESHANDLING



$cattles : \mathbb{P}\, Animal$
$adult : \mathbb{P}\, Adult$
$calfs : \mathbb{P}\, Calf$

$\{cattles, adult, calfs\} \in attributes$
$cattles = adult \cup calfs$

sellCalfs

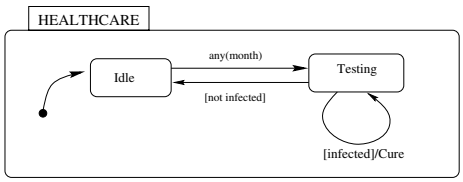
$\Delta calfs$

$\#calfs' \leq \#calfs$

HealthCare

$Role[HEALTHCARE/behavior]$

HEALTHCARE



$infected : \mathbb{P}\, Animal$
$reminiscence : \mathbb{N}$

Cure

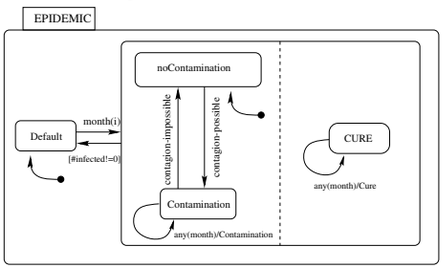
$\Delta reminiscence$

$\#infected \neq 0 \land reminiscence' = 3$

Anademie

$Role[EPIDEMIC/behavior]$

EPIDEMIC



$cattles, sanes, infected : \mathbb{P}\, Animal$

$cattles = sanes \cup infected$

Cure

$\Delta(infected, sanes)$

$\#infected - \#infected' = \#sanes' - \#sane]$

Contamination

$\Delta(infected, sanes)$

$\#sanes - \#sanes' = \#infected' - \#infected$

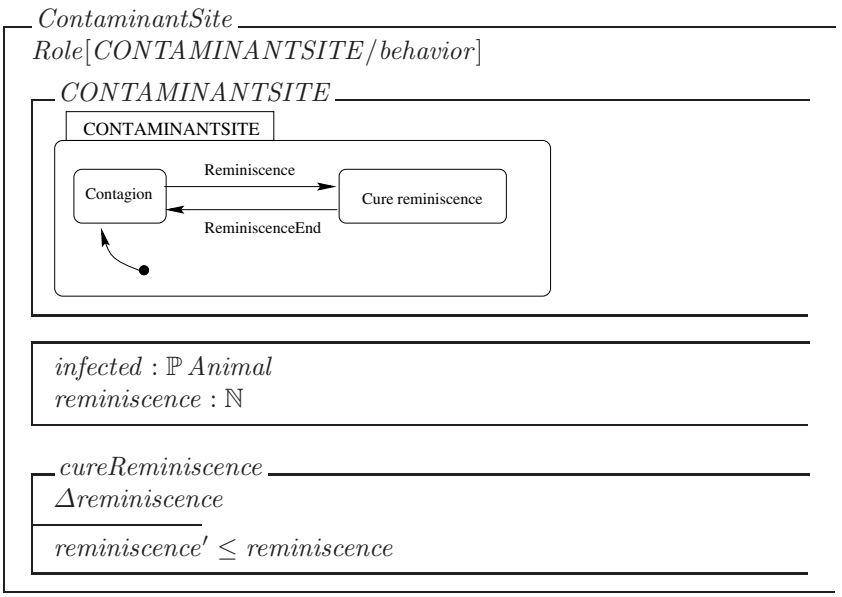
3.2 Specification Analysis

The analysis is performed by using STATEMATE [14]; an environment which allows the prototyping and the simulation of the statechart specifications. The specification analysis is based upon execution of the statecharts and can be done using two techniques.

The first technique is *simulation* and the second is *animation*. In our case simulation would consist in assigning probabilities to events or actions occurrences. With this technique one can evaluate quantitative parameters of the specified system.

For the Aphtous Ulcer Fever example probabilities can be associated to calfs selling or birth so one can evaluate how many animals are involved in each operation. Probabilities can be useful in different disease steps to simulate curing or infections of sane animals. Animation technique consists of testing the specification with predefined interaction scenarios. It enables one to test if the system behavior is consistent with requirements. For example, we have tested that if there is at least one infected animal the Health Care role executes *Cure* operation.

The simulation tool offers an interactive simulation mode and a program controlled mode. In the latter a program written in a high level language replaces the user. One feature of this programming language is the breakpoint construct. Breakpoint stop the specification execution when a condition is verified. Possible uses of breakpoints are, for example, configuration tests with predefined interaction scenarios and output of statistics. For the former one can test if all role reactions correspond to months schedule. For the latter one can associate probabilities for different illness phase so one can simulate illness evolution. The figure 2 is the result of such simulation. On x-axis we have month of the simulation and on y-axis animals count. The two curbs are for Sane and Infected animals. One may see that each time animals are at farm the number of infected animals augment and then drop when they return at field. Indeed, contagion is only possible when animals are at farm.



4 Related Work and Conclusion

In this paper, we have presented a formal specification approach for MAS based upon an organizational model. The organizational model describes interaction patterns which are composed of roles. When playing these roles, agents instantiate interaction patterns. For the sake of clarity we have only presented roles specification. But there are an agent class in our framework which enables the specification of evolving behaviors. This model is well suited for describing complex interactions which are among MAS main features. Furthermore, each model concepts are given a formal semantic within our specification framework. The

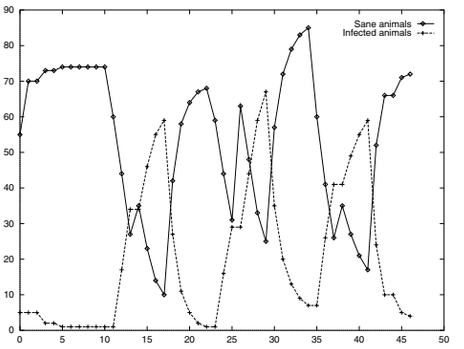


Fig. 2. Statistics of Aphtous Ulcer Fever simulation

language used by the specification framework can describe reactive and functional aspects. It is structured as a classes hierarchy so one can inherit from these classes to produce its own specification. The used specification language allows prototyping of specification. Prototyping is not the only means of analysis, indeed, in another work [9], we have introduced a formal verification approach. Moreover, the specification structure enables incremental and modular validation and verification through its decomposition. Eventually, such a specification can be refined to an implementation with multi-agent development platform like MadKit [11] which are based upon an organizational model [7].

Formal theories are numerous in the MAS area but they are not all related to concrete computational models [4]. Temporal modal logic, for example, have been widely used [19]. Despite the important contribution of these works to a solid underlying foundation for MAS, no methodological guidelines are provided concerning the specification process and how an implementation can be derived. Another type of approach consists in using traditional software engineering formalisms [16]. Our approach is of the latter type. Among these approaches a few [16,2] provide the specifier with constructs for MAS decomposition. The approach of Luck and d'Inverno has two drawbacks: the use of Z make MAS reactive aspects difficult to specify [8] and these specifications aren't executable so simulation and prototyping are not possible. The formalism used in DESIRE also lack the support regarding simulation or prototyping.

Despite the encouraging results already achieved, we are aware that our approach still has some limitations. Indeed, it doesn't tackle all problems raised by MAS development. But we believe that this first step constitutes a specification kernel leading towards a practical approach to formal specification of MAS. Among issues remaining for future work the organizational model used by us needs more work to do ahead. The play-by relationship of our example is of the simplest type. In order to deal with more complex case we have to explore the semantic of this relationship. Moreover, the Object-Z part of the specification is

not yet executable. However a preliminary work [10] has shown that it is possible to give an operational semantic to Object-Z but it must be strengthened.

References

1. Federico Bergenti and Agostino Poggi. Exploiting UML in the Design of Multi-Agent Systems. In *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2000. 117
2. F. M. T. Brazier, B. Dunin Keplicz, N. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997. 125
3. Cristiano Castelfranchi. Engineering Social Order. In *Engineering Societies in the Agents' World*, Lecture Notes in Artificial Intelligence. Springer Verlag, 2000. 116
4. Mark d'Inverno, Michael Fisher, Alessio Lomuscio, Michael Luck, Maarten de Rijke, Mark Ryan, and Michael Wooldridge. Formalisms for multi-agent systems. In *First UK Workshop on Foundations of Multi-Agent Systems*, 1996. 114, 125
5. Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Departement of Computer Science, University of Queensland, AUSTRALIA, 1991. 115
6. Benoit Durand. *Simulation multi-agents et épidémiologie opérationnelle*. Thèse de doctorat, Université de Caen, 1996. 115, 116
7. Jacques Ferber and Olivier Gutknecht. Aalaadin: a meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS'98*, july 1998. 115, 125
8. M. Fisher. If Z is the answer, what could the question possibly be? In *Intelligent Agents III*, number 1193 in Lecture Note of Artificial Intelligence, 1997. 125
9. Pablo Gruer, Vincent Hilaire, and Abder Koukam. an Approach to the Verification of Multi-Agent Systems. In *International Conference on Multi Agent Systems*. UTBM/SET, IEEE Computer Society Press, 2000. 125
10. Pablo Gruer, Vincent Hilaire, and Abder Koukam. Verification of Object-Z Specifications by using Transition Systems. In *Fundamental Aspects of Software Engineering*, number 1783 in Lecture Notes in Computer Science. Springer Verlag, 2000. 126
11. Olivier Gutknecht and Jacques Ferber. Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. Technical Report 97188, LIRMM, 1997. 125
12. D. Harel and A. Pnueli. On the development of reactive systems. In K. R. Apt, Editor, *Logics and Models of Concurrent Systems*. Springer Verlag, 1985. 115
13. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987. 115
14. David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michal Politi, Rivi Sherman, Aharon Shtull-Trauring, and Mark B. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990. 123
15. T. Lissajoux, V. Hilaire, A. Koukam, and A. Caminada. Genetic Algorithms as Prototyping Tools for Multi-Agent Systems: Application to the Antenna Parameter Setting Problem. In Springer Verlag, editor, *Lecture Notes in Artificial Intelligence*, number 1437 in LNAI, 1998. 115

16. Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In AAAI Press/MIT Press, editor, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260, 1995. 125
17. Richard F. Paige. A meta-method for formal method integration. In John Fitzgerald, Cliff B. Jones, and Peter Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 473–494. Springer-Verlag, September 1997. ISBN 3-540-63533-5. 117
18. Trygve Reenskaug. *Working with Objects: The OOram Software Engineering Method*. Manning Publications, 1996. 116
19. Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. Available by FTP, 1994. Submitted to The Knowledge Engineering Review, 1995. 125
20. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, Editors, *Proceedings of the Third Annual Conference on Autonomous Agents (AGENTS-99)*, pages 69–76, New York, May 1–5 1999. ACM Press. 115
21. Pamela Zave and Michael Jackson. Conjunction as composition. *acm Transactions of Software Engineering and Methodology*, 2(4):379–411, October 1993. 117

Combining Software Components and Mobile Agents*

Mercedes Amor, Mónica Pinto, Lidia Fuentes, and José María Troya

Depto. de Lenguajes y Ciencias de la Computación. Universidad de Málaga
Campus de Teatinos, s/n E29071-Málaga,Spain
{pinilla,pinto,lff,troya}@lcc.uma.es

Abstract. We present a first approach that combines the mobile agent and the compositional paradigms into a new agent-based compositional model. The aim of this work is to explore the capabilities of both paradigms in improving the design and development of open and distributed systems. Our goal is to add compositional characteristics to mobile agents by defining an agent-based compositional model. We take advantage of the mobility and the composition capability by applying this model to the problem of searching software components in partially-instantiated applications, both in design and run time. That is, the abstract components of an architectural design of an application can be fulfilled with concrete ones by retrieving them from the Internet.

1 Introduction

Nowadays, as a result of the growing use of the Web and networks, the complexity of software systems is increasing. Due to the heterogeneous nature of the distributed environments where they should run, these systems need to model new kinds of interactions, and also they may be able to be adapted to changing requirements and new trends. In order to cope with this complexity we need new paradigms that facilitate the design of distributed and open systems in a better way than the object-oriented paradigm.

Having into account actual running trends of the software engineering, two paradigms are applicant for this purpose, the component and agent-based paradigms. Both of them seem to represent an evolution of the object-oriented paradigm. However, each paradigm focuses on different aspects of distributed applications and so they might be complementary.

The mobile agent paradigm is especially attractive to perform complex, tedious or repetitive tasks in open and dynamic systems [1]. Agent-based applications are develop specially for dynamic, distributed, open and heterogeneous environments, such as the Internet. The basic entity for designing and building this kind of applications is the agent, which can be seen at some scale, like an active object.

* This research was funded in part by the CICYT under grant TIC99-1083-C02-01, and also by the telecommunication organization "*Fundación Retevisión*".

A. Omicini, R. Tolksdorf, and F. Zambonelli (Eds.): ESAW 2000, LNAI 1972, pp. 128-142, 2000.

Therefore, the mobile agent paradigm may represent another step in the evolution of the object-oriented paradigm [2].

Although, a mobile agent may be viewed as an object, because it has a state, behaviour and identity, and also it has location, that may change during its lifetime. Due to this feature, mobile agent paradigm represents a design pattern, which is considered useful to develop dynamic and distributed applications. Some of the main benefits that mobile agents offer in distributed environments are [3]: they improve system performance, because they reduce the network load, and they let to overcome network latency in critical real-time systems. Also they encapsulate protocols that can easily evolve to new requirements. Although current trends on Internet computing lead to the use of mobile agents, mainly to perform searching tasks, distributed information retrieval and personal assistance, some technical and non-technical hurdles remain yet, since in many aspects is still a novel paradigm [4]. Some of these hurdles might be clarified before mobile agents could be widely used.

The component-oriented paradigm is also considered as another step in the evolution of the object-oriented paradigm. Objects and components both make their services available through standard interfaces. But the software component definition covers also aspects such as semantic of composition, introspection, binary interfaces, and also, component market related aspects such as reutilization of third parties components (COTS) and the independent deployment issue [5]. Component paradigm is gaining acceptance as the demand of assembling systems using components increase the productivity and decrease the cost of developing software, representing in some manner, the industrialisation of software development [6].

Therefore, both these paradigms exhibit features that seem to make them appropriate for designing complex systems in an easy way. Mobile agents might be an effective choice as they provide a single, general framework in which distributed applications can be implemented efficiently and easily, but present some drawbacks [4]. Our goal is to define a new model taking advantage of the features of both paradigms for designing open distributed systems.

We have a wide experience in componentware developing MultiTEL [7], a compositional framework of the domain of multimedia and collaborative services. But we have found that some of the agents' characteristics such as the mobility and autonomy could endow more flexibility to the component-based design. The architecture of the MultiTEL framework follows a compositional model, which defines standard components and common patterns of user interactions as connectors from the multimedia services domain. Once the coordination patterns have been identified, these can be tailored to obtain new services also reusing components, thus decreasing the development time of multimedia services.

In this first approach, we present an agent-based compositional model, which has components and compositional agents. We have defined the MultiTEL2 platform that provides a distributed environment for supporting the execution of applications based in the agent-based compositional model. By merging both paradigms, we aim to design more open application architectures, facilitating the construction of customisable applications over a distributed platform and providing an open market of components and agents-off-the-shell (like COTS).

One of the main contributions of our work is that we have applied this model to develop and bind partially-instantiated applications, where abstract components can

be dynamically instantiated with real software components that can be obtained at runtime from the net. Each abstract component has the ability of a search mobile agent encapsulating the searching criteria (e.g. incoming and outgoing messages defined as part of a component interface) for searching plug-ins in specific locations. An interesting and real example that can take advantage of this feature is retrieving a the software component that models a network camera connected to the Internet and managed from an unknown host. This kind of camera has a with built-in web server, and its own IP address, which means that it is not plugged to a host for the capture and transfer of images. Therefore, the component can be at any host inside the network. By this way, at runtime, a mobile agent obtains this component, and the network camera can be used inside the application like a local device, without a previous knowledge about where is the network camera or which is its IP address.

2 Components vs. Agents

At present, software engineer experts are discussing about the issue "agents versus objects". However, we consider that this discussion should be moved to the component field, and confront agents and components, since we consider that both paradigms are at the same level of abstraction and also can be implemented using the object-oriented paradigm. Firstly, we are going to discuss what components may offer to agents.

Getting a look to the literature about components, we cannot find a consensus about what are the characteristics that a component might offer, but there are some of them that are present in almost every component definition. *Introspection* lets components to 'say something' about their behaviour and data. This property, which is not found on current agent systems[18], would allow agents to interoperate dynamically, and it would let them to afford, for example, a protocol based on introspection, without a previous knowledge of the peer agent's public IDL.

Dynamic composition means that components can be linked at runtime, and it is possible if components provide some information about them. This information must be included in components IDLs, which describe not only their incoming messages but also their outgoing messages [8]. Agents also show this property, as they can communicate with other agents and their environment, but in a predetermined way. An agent does not know who or when are they going to communicate, but they know how. For example, a search agent, which interacts to a predetermined kind of database, knows how to interact (the consult and the result are given in a well-known format). There is no flexibility. If the syntax changes or if it wants to dialogue with unknown and third party agents, the agent is not able to adapt its behaviour to it. Agent-based designs could be more flexible if the interaction is determined at runtime, when agents meet and after a negotiation about establish how they are going to interact.

Other property of components, is the *event-handling*. This mechanism affords a kind of dynamic composition via an anonymous interaction. The event sender does not need to know who could be the receiver of the event, which is established by composition. This mechanism offers flexibility to agent models, as an agent request

can be handled like an event received from the environment. That is, if an agent needs a service, it throws a request, which will be caught by other unknown agent. Some agent communication systems based on tuple spaces support a kind of anonymous interaction to coordinate mobile agents [9][17]. However, at componentware field this has greater semantic load because the matching between input and output requests is established according to an architectural pattern.

In addition, components must have explicit context dependencies and have to provide binary interfaces, that is, a component might be delivered in a binary form to be ready for use without need of exploration of the implementation code. Furthermore, a component should be deployed independently, because software components will be sell inside a global component market. To carry out this issue, component needs to have explicit and well-documented context dependencies in order to be reused in any environment. Agents should also present these properties, as it would be possible to provide an open agent market, where software developers can make use of agent-based solution and reuse them. The benefits would be greater if agent's developers would come to an agreement about standards like agent communication language, agent mobility, and so on.

On the other hand, the agent paradigm has become an increasing and important area of software engineering research. Application domains in which agent-based solutions are being applied and investigated fill different disciplines, such as AI, Genetics, Robotics; but agents also are considered a promising technology for the design and development of distributed systems [10]. In distributed and open systems, executable code travels with mobile agents so an agent can be considered as a mobile object enriched with other features [11]. The goal of agents is to delegate and automate tasks, most of them tedious, in an asynchronous way. That is the reason why agents are being applied to the Web, for instance in searching and filtering information, intelligent electronic mail, electronic commerce, mobile computing and telecommunications [12]. We consider that the main characteristic that differentiates the agents from other design paradigms, included compositional paradigm, is its autonomy [13].

Agents are autonomous, meaning that they act independently in order to perform their responsibilities, although an agent may interact with others. In addition, it can control how and when it answers to requests, and it has the ability to react to changing conditions in the environment without receiving a specific request. Others agent's attributes that could characterise an agent are: the mobility, whether it can move from a host to another; intelligence, if it can reason; adaptability, if it can learn. Also, agents can have knowledge about some domain; continuity, persisting over time. Unlike, components have a meaning inside the context of a domain-specific framework. The reutilization of a standalone component is not practical, it is desired to reuse a component inside a compositional framework, as part of a task. However, components are passive entities that only act when they receive input messages or events, and its behaviour depends on them. Agent's autonomy makes a component to be an active entity.

In the next section, we present a first approach that merge and integrate both paradigms, to obtain a new agent-based compositional model. Our aim is that applications derived from this model can take advantage of both paradigms.

3 The Agent-Based Compositional Model

The proposed agent-based compositional model is based on the component model defined by MultiTEL [7]. Since we combine agents with components our agent-component model has two different types of entities: components and c-agents. Components model real entities of the system and encapsulate computation, while c-agents are active entities oriented to a task. This proposed model should be applied in the design phase of a distributed application.

3.1 Components and C-Agents

Components are passive entities that model resources and real world entities like a camera, a printer, or a database. The IDL of a component defines the list of messages that can be received or sent by the component. The reception of a message causes changes in the internal state of components that are reported to the environment by sending one or more messages specified as part of its IDL. This means that an IDL does not only include information about the incoming messages as usual and also the outgoing ones.

Components have an identity independent of the different interactions in which they could engage. They do not know how the reception of a message and its computation influence the rest of the system. This characteristic contributes to have a component market from third parties (COTS) where components can be reused in new services without knowing their implementation, only knowing their interfaces, studying the services that they offers and the messages that they may send. A component may define private components for modularization purposes, and in this case must include in its IDL the subcomponents' incoming and outgoing messages. Incoming ones are delegated to the suitable subcomponent.

C-agents are autonomous active mobile entities, which are in charge of a task. C-agents can navigate through a set of places. In opposite with components, c-agents know how the reception of a message and its computation influence the rest of the system, and also indicate some information about the receiver of an emitted message. C-agents also provide an IDL that specifies the incoming and outgoing messages. Apart from the dialogue that may be established between c-agents and components, c-agents can communicate between them through the standard XML language.

Both, components and c-agents, have a unique identity, like a URL, assigned on creation, which lets distinguish itself from others components and c-agents. Also, components and c-agents are characterised by a role. For components, the role is an identifier that represents the resource that models i.e. the component *camera* models a camera device. For c-agents, the role represents a set of abilities, i.e. *info-search* c-agent, it is a c-agent that can search information. The role of components and c-agents is useful in the context of application architecture, because by publishing the role, a component advertises the resource that models or represent, and a c-agent advertises that it can perform a collection of operations, tasks or services. Many components or c-agents with the same role can be running inside the application. In some way, the role of a component or of a c-agent is characterised by its IDL, that is, its outgoing and incoming messages. A component *printer* will contain in its incoming messages

something like *print_job*, or *ready*, related with a printer device. All components that play the same role or have the same functionality should implement IDLs containing the required input and output messages. Likewise, a c-agent with the role *search* will receive message to search for, and emit messages to return a search result.

3.2 Interaction: Message Delivery

Communication between components and c-agents is established by message passing. Since components are passive entities they only react to c-agents requests or they ask unknown c-agents for a service by a message in an asynchronous way.

As shown in

Figure **1**, messages enclose the identifiers of the sender (from) and the recipient (to).

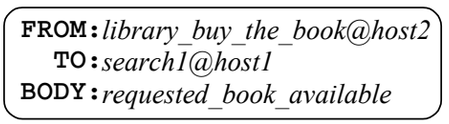


Figure 1. Message format

However, the field to that contains the recipient of the message does not need to have a reference of a component or c-agent identifier. There are three way of specifying the component or c-agent target of the message. Depending on the information about the recipient attached to the outgoing messages, the model defines three types of message delivery:

- *Role-based*. The recipient of the message can specify receiver's role. For example, a c-agent with the address *search1@host1* is searching among digital libraries for a certain book. It has just arrived to a new host and wants to send a message to components *books_ database*, asking for the book. In this case, the c-agent does not have to know the identifier of the local component, but it knows the role that characterises the component. Also, since more that one component *book_database* can be host at the same location, the message will be delivered to all.

- *Identity-based*. The receiver's identifier is specified when the sender emits the message. Component and c-agents can obtain the identifier of another component or c-agent from a previous incoming message, which allows answering a request. Following the previous example, the *books_database* component *library_buy_the_book@host2* can send an answer message to the c-agent *search1@host1* address, notifying that the book was available.

- *Content-based*. When any information about the recipient is given with the message, the recipient or recipients of the message will be all the components and c-agents whose IDL contain the message, that is, every component that offer the required service. For example, a component that models a printer can send a message to *anybody*, without putting an address, an identifier or a role in the field TO of the

message, indicating that it is ready. This means that the target c-agents or components do not have to play the same role, only need to include the required message.

On the other hand, communication between c-agents is defined in a different manner, since it involves protocol specification and also c-agents can take part in more complex interactions, like negotiation, complex queries, or exchange information. In order to model the complexity and the heterogeneity of message formats c-agents talk to one another by specifying their requests in XML, a common understood language. By XML we can describe a protocol, a data structure or a script. By this way a c-agent can define a protocol, or it may send a message not contained in the IDL of the target c-agent and interchange information represented in an independent format letting to describe any data content. In addition, XML lets to define new labels and attributes. For example, a c-agent has a search result that is stored in a data structure, and this result is requested by another c-agent. The interchange will be performed expressing the data structure and its content by XML, as the target c-agent does not have to know how it is the internal data structure of the other c-agent. Also, the c-agent can request the result, including the required format of the information that has to be returned.

XML has enough expressiveness to specify the IDL of both a c-agent or a component. Figure 2 shows the XML's specification of the c-agent *Cam_Search,* by the description of its incoming messages (*ListResult*) but also its outgoing messages. By a XML document, a c-agent can inform about its public IDL, that is known as introspection. On current agents' systems [18] if an agent wants to send a message, it has to know previously the other's public IDL, because there is no way to get it at execution time. Furthermore, if the implementation language offers a mechanism to discover the public IDL, as reflection in Java, it is not given in a standard format and it depends on the implementation language and the platform, that does not facilitate agents interoperability. However, introspection and the use of standard XML allow c-agents to know the public IDL of others c-agents, at runtime, to start a communication. There is no need of a previous knowledge of each other, and XML provides with a standard format, that makes easier c-agents interoperability.

```
<c-agent><c-agent.role>Cam_Search</c-agent.role>
<c-agent.implementation>MyCamSearchv1</c-agent.implementation>
<c-agent.IDL>
<c-agent.IDL.Incoming>
<c-agent.IDL.Incoming.Message>
<c-agent.IDL.Incoming.Message.body>ListResult</cagent.IDL.Incoming.Message.body>
<c-agent.IDL.Incoming.Message.param>cameralist</c-agent.IDL.Incoming.Message.param>
</c-agent.IDL.Incoming.Message>
</c-agent.IDL.Incoming>
<c-agent.IDL.Outgoing>....</c-agent.IDL.Outgoing>
</c-agent.IDL>
</c-agent>
```

Figure 2. XML specification of a c-agent's IDL

Furthermore, we can use the XML language as an abstract notation to specify the messages that c-agents interchange. That is, the communication of c-agents can be performed by exchanging the messages of a communication protocol in XML format, instead of, for example, method invocation. By this way, once a c-agent receives the message formatted in XML, it parses and interprets, and

sends the corresponding response also in XML. The use of XML ensures a high degree of interoperability with others agents' platforms that support XML as a communication content language [18].

4 MultiTEL2 Platform

MultiTEL2 is a middleware platform that offers the services required for the composition of applications based on the model defined in last section, and provide support for mobile agent characteristics of c-agents. Components and c-agents will run on a distributed platform that supports the communication mechanisms based on message delivery, defined by the model. MultiTEL2 provides common services for the distributed execution of applications and lets components and c-agents coexist and cooperate. Therefore MultiTEL2 is a development and runtime platform supporting applications based on the agent-based compositional model.

Figure 3 shows the MultiTEL2 platform, which provides a Component and C-Agent Creation Service, Naming & Locating Service, a Distributed Component Repository and a C-Agent Migration Facility that will be explained in the following sections. The platform has to fulfill the tasks of supporting creation, execution, localization, migration and communication of c-agents and components.

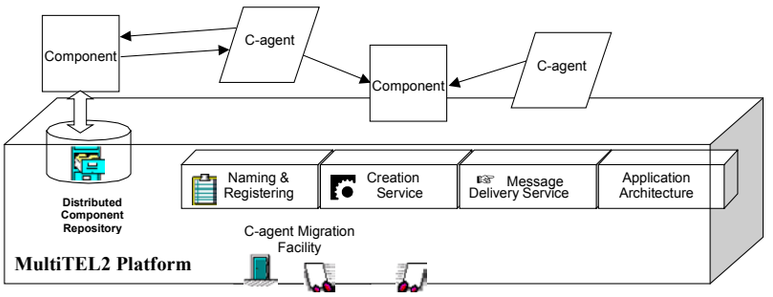


Figure 3. MultiTEL2 platform

The platform and the agent-based compositional model that support migration are developed in Java. Thus, the middleware platform can be installed on any computer that provides a Java runtime environment seeing the components and c-agents platform independent. In addition, the migration implemented, by transferring the bytecode with the Java class and the object state, removes the need to recompile the agent on arrival at a new host

4.1 Application Architecture

Standalone components and c-agents are useful when they take part of an application. In order to define the architecture of an application based on components and c-agents, it is needed to specify the requirements imposed to components and c-agents to take part of an application. These requirements are expressed in terms of the role

and the incoming and outgoing messages. The data structure that holds the application architecture is maintained at the MultiTEL2 platform, and it can be consulted by the platform services.

This structure shows how can be performed the composition between components and c-agents. For example, the application that buys a book has to be composed by c-agents *info-search* and components *book database*. Also, some of the incoming messages of the component need to appear as outgoing messages in the c-agents IDL and vice versa.

The architecture can be defined and stored in the data structure shown in

Figure **4**, where, for each component and c-agent that participates in the application, it is determined its role, the set of its incoming and outgoing messages, and its implementation. Also, for each c-agent the architecture specifies the components and c-agents that it may interact.

Role	Implementation	Incoming messages	Outgoing messages	Related to
Book_database	Oracle_library	Database_input	Database_output	
Inf_Search	Book_searcherv1	Search_request	Search_result	Book_database
Inf_Search	Book_searcherv2	Search_request	Search_result	Book_database

Figure 4. Application architecture

4.2 Component and C-Agent Creation Service

This service attends requests from c-agents to create components and c-agents. The requests must specify the role of the component or the c-agent that have to be created, i.e. *create_component(camera)*, or *create_c-agent(search)*. The platform creates a component or a c-agent using the description given the application architecture, shown in Figure 2, where this service can map an implementation to the requested role. Components and c-agents are created dynamically along application execution.

4.3 Naming and Locating Service

The components and c-agents must have an own unique identifier to be univocally addressed. Once they are created, under a role inside an application, the platform assigns them an identifier. The service includes the host where it has been created as part of its identifier. The host where a c-agent is created is also known as the c-agent's home.

C-agents are mobile, which means that it can move through different hosts, and its location can change continuously. As we explained in last section, message delivery is determined by recipient, not by location, i.e. when a component sends a message to a c-agent, it only has to specify the c-agent's identifier, so the platform needs to maintain a data structure to register the last location of a c-agent. The actual location of a c-agent is updated at c-agent's home.

Figure 5 shows how the localisation of remote c-agents is performed. When the platform is requested to send a message (e.g. to "*Search1@host1*"), first it checks if the c-agent is running locally, consulting local context. If this checking is negative, the platform asks for the current location of the c-agent's home, in this case "*host 1*"

(step 1). The c-agent's home consults a data structure (step 2) where actual location, which is "*host 3*", is registered, and returns that information (step 3). After receiving the actual,"*host 2*" will remit the message (step 4). Every time the c-agent moves to a new location, its home registers the new c-agent's location.

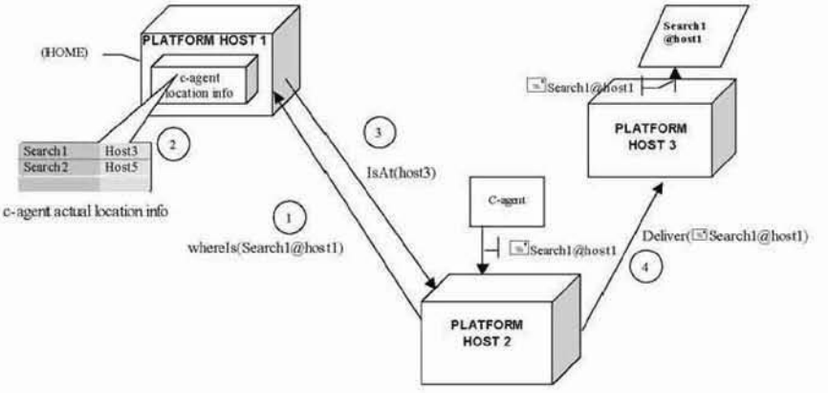


Figure 5. Localisation of mobile c-agents

For every c-agent created on a host, the platform maintains a structure like it is shown in Figure 3 as *c-agent actual location info*, to maintain a c-agent migration registry for delivering messages to moving c-agents in a transparent way. This is not necessary for components, as they are not mobile. However, the number of emote messages might be the less as possible, and the interaction should be local, as one of the advantages of the c-agent's mobility is that the need for network availability is reduced to the short time of its migration.

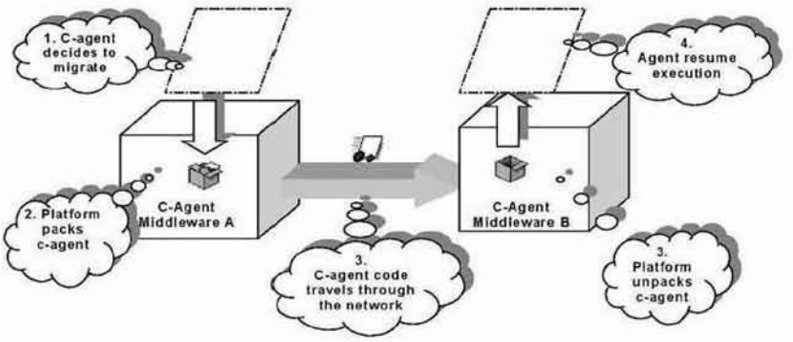


Figure 6. How Agent Migration is performed

4.4 Agent Migration Service

The platform is able to freeze an executing c-agent, and transfer its code and its state to another host. Moreover, the platform is able to unfreeze a c-agent transferred from another host and allow it to resume execution as shown in Figure 6. The c-agent migration can be easily implemented using Java, through object serialization and reflexive programming. Due to this strong migration, the target host does not need to have the Java class that model the c-agent.

When a c-agent wants to migrate, the platform deals with notifying c-agent's home that it is going to move (step 1 in Figure 6). Then, it takes care of packaging the agent, including its actual state (step 2), and sends it through the network to the destination host (step 3), which is in charge of unpacking the c-agent, and registering it into the local context (step 4). Then it notifies c-agent's home the new location and resumes its execution from its last state (step 5).

4.5 Message Delivery Service

Communication is performed by asynchronous message passing. The platform receives messages and delivers them to the right component or c-agent, like a post office. To deliver a role-based message, the platform first consults the local context, that is, the components and agents that are running locally and that fulfil the requested role. If there is no running component or c-agent fitting the role, it creates one.

Identity-based messages are delivered directly to the right component and c-agent. If they are not running in the local context, the platform deals with the remote delivery. In order to prevent message lost during migration, c-agent's home will queue messages. Once the migration is completed, the platform notifies it to its home host and then the home will dispatch the received messages to the c-agent at its new location.

Content-based messages are delivered to local components and c-agents. The platform only looks for components and c-agents that can receive the message, because it is contained in their incoming messages and also, are running locally. If the service does not find any applicant component or c-agent the message will not be delivered.

4.6 Distributed Component Repository

MultiTEL2 enforces to share and reuse components and c-agents. To make easier this task, the platform offers a Distributed Component Repository (DCR). In this way, application designers may reuse public software developed by others companies that conforms MultiTEL2 framework principles.

Designers may reuse a component or a c-agent in different contexts, so it is needed to provide information about its features and capabilities. This metadata, generally implemented in terms of IDL [14], shows component and c-agent's behaviour, and will help to classify their features easing their reuse in other contexts.

Following Web principle of information distribution, every MultiTEL2 machine has a Component Repository, organised in hierarchical domains. Each entry of the repository corresponds to an implementation of a component or a c-agent, and contains some information needed to know component and c-agent behaviour. The information available for a component is its IDL and a brief textual description of its behaviour. In the c-agents case, also include the task they can develop. For both we also include the assigned role that the component or the c-agent may perform in any service architecture.

5 An Agent-Based Solution for Retrieving and Reusing Software Components

Classification and selection of reusable components are considered as a key success factor. Software component suppliers should provide standard documentation, to help customers in finding quickly the right component.

But, instead of performing an individual and manual search, we propose to delegate this task to c-agents. That is one of the purposes of c-agents: search and retrieval components and c-agents from DCR. The need of searching a component or a c-agent can arise at two stages of the software developing process, at design time and at run time. A kind of c-agents called *DCR_Search* will bring the desired software component, directly to the MultiTEL2 platform. These are basic c-agents that perform a search throw the repository and are included as part of the Visual Design Tool (VDT) for being used at design time (Figure 7).

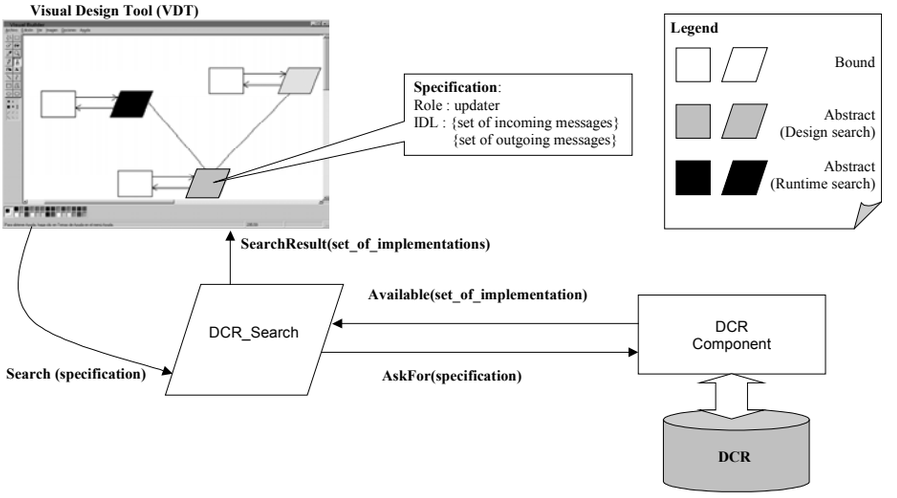


Figure 7. Agent-based search of a component at design time

As we explain in a previous section, application architecture is defined specifying components and c-agents' roles, and connections between them. The developer may

design an application architecture by using a VDT, inserting components and c-agents, specifying the role they must perform inside the application and the connections between them. After that, the developer must choose an implementation that conforms all these requirements. Instead of implementing every component and c-agent, the developer can reuse other implementations, searching and consulting the components and c-agents available in the DCR.

But, instead of searching manually through the DCR, this task is performed automatically by a kind of c-agent called *DCR_Search*, shown in Figure 5. It carries out the requirement, that is, the role, the incoming and outgoing messages or a combination of both. The *grey hole* means that it corresponds to an abstract component or c-agent which implementation is not set yet, so the c-agent of the VDT, carrying the specification, searches for a suitable implementation to fill in the *hole*, saving developer's time. Once the c-agent finishes its search and presents its results, the developer will decide which component or c-agent will be used in the architecture by inspecting the information about the applicants retrieved by the c-agent.

We go further away, and we propose to delegate the searching of components or c-agents implementation until run time. Thus, the application will run over a partial-instantiated architecture, like a template, where the field implementation of some components or c-agents can be empty (in

Figure **4**) or a *black hole* of the VDT in Figure 7. When the application running on MultiTEL2 needs to instantiated a component or a c-agent, but its implementation is not specified in the application architecture, a kind of c-agent called *creator* is instantiated instead. This c-agent is a specialisation of the searcher c-agent used above, because it searches and also it takes the decision about the suitable implementation inside the result set. Once the c-agent finishes its search, it returns to the original host and creates an instance of the chosen component or c-agent. The user will not be aware of that different running instances of the application may have different component or c-agent implementations, since the c-agent search result can differ, but the restrictions imposed by the application architecture would be enough to guarantee that the implementation chosen by the *creator* c-agent at runtime will suit into the architecture requirements. This tool would let to perform a rapid prototyping of distributed applications based on the proposed model, including the developing of components in very short time, and with less effort. In workshop discussion , people suggested that this capability is no present at current tools.

In addition, we can take another advantage from letting the application with an "incomplete" design. Let us take a provider of a product that has to be distributed to different vendors, which want to be notified about the last updates of the product. Instead of taking care of notifying the updates the provider puts the last version of every component and c-agent in its own DCR.

On the other side, we can take advantage of this solution to the provision of customisable software products. Different vendors provide the same product but they could be able to customise the component that provides the user interface. With our approach, the developer does not need to worry about it, leaving the decision about what implementation will be used until runtime. Every time the component is required, it will be found at the nearest DCR, for instance, the local DCR of the provider. By this way, the same application could display different graphic interfaces or whatever, depending on the vendor the user is subscribed.

From the given solution, the applications frameworks partially instantiated, we can find another utility, very fashionable nowadays. The run time retrieval of software can be applied to plug-in multimedia devices without worrying about download the needed drivers. When a multimedia application needs to create a component that manages the camera, only has to look for the best implementation for that camera model. This runtime search can be applied to any multimedia device component, but also to any system property that could enhance the performance of the application.

6 Conclusions and Future Works

In this first approach, we have presented an agent-based compositional model that combines mobile agent and compositional paradigms. We have obtained benefits from merging both paradigms that may improve the design of open and distributed systems, especially those that are implemented above the Web. For instance, by applying mobile agent characteristics we can improve distributed searching tasks, and with componentware applications we can improve independent software components. By endowing the agent-based design with compositional issues, we have tried to make it a more flexible and powerful paradigm applicable to any distributed system. One of the most important contributions of this work is that due to the compositional characteristics of the model, users can develop partially-instantiated applications based on components and c-agents, which could be dynamically instantiated at runtime. The benefits are reflected directly on the final software solution as it can provide customisable plug-ins, or the latest version available, every time the application is executed. The result is a dynamic and evolving application, that adapts itself dynamically to new user requirements and the last available technology.

Our future goals are to complete the definition of the agent-based compositional model presented in this paper, and to specify it formally[15]. A further goal of our research is to address the c-agents interoperability issue[16], and add the introspection property to c-agents by using XML, which allow the developing of flexible and reusable societies of heterogeneous c-agents that would interact using XML standard. Another goal is to build a working prototype of the platform and proposed the tool, regarding open distributed systems features like scalability, security issues, fault tolerance and interoperability with existing agents systems[18].

Furthermore, we will study how to incorporate additional characteristics that emerged from the workshop discussion, such as integration with mobile agent's standard, or to include a executable representation language for specifying c-agents.

References

[1] A. Silva, J. Delgado, "The Agent Pattern for Mobile Agent Systems". *3rd European Conference on Pattern Languages of Programming and Computing*. 1998.

[2] B. Venners, "Solver Real Problems with aglets, a type of mobile agents". *JavaWorld*. http://www.javaworld.com/javaworld/jw-05-1997/jw-05-agents.html. May 1997.
[3] D. Lange, M. Oshima, "Seven good reason for Mobile Agents". *Communications of the ACM,* Vol. 42, No.3 pp. 88-89. March 1999.
[4] D. Kotz, R. Gray, "Mobile Code: The Future of the Internet". *MAC3 on Autonomous Agents'99*. http://mobility.lboro.ac.uk/MAC3/. May 1999.
[5] C. Syperski, "Component Software. Beyond Object-Oriented Programming". *Addison-Wesley*, Reading Mass. 1998.
[6] F. Slootman, "A blueprint for Component-Based Applications". *Compuware Corp*. http://www.compuware.com/products/uniface/station/reading/ind_blue.htm.
[7] L. Fuentes, J. M. Troya, "MultiTel: Multimedia Telecommunication Services Framework". A chapter of the book *Domain Specific Application Frameworks*. Wiley & Sons. October 1999.
[8] Á. Ólafsson, D. Bryan, "On the need for required interfaces of components". *Special Issues in Object-Oriented Programming*, pp 159-165. Verlag Heidelberg. 1997.
[9] A. Omicini, F. Zambonelli, "Coordination of Mobile Agents for Information Systems: the TuCSoN Model". *6th AI*IA Convention*. 1998.
[10] H.S. Nwana, "Software Agents: An Overview". *Knowledge Engineering Review*, Vol. II, Nº 3, pp.1-40. Cambridge University Press. September 1996.
[11] D. Chess, et al., "Mobile Agents: Are They a Good Idea". *IBM Research Report*, 1994.
[12] D. Chess, et al., "Itinerant Agent for Mobile Computing". *IBM Research Report*, 1995.
[13] D. Kafura, J.P. Briot, "Actors and Agents". *IEEE Concurrency*, pp. 24-29. April-June 1998.
[14] D. Krieger, R. Adler, "The Emergence of Distributed Component Platforms". *IEEE Computer*. March 1998.
[15] R. Bergenti, A. Poggi,"Exploiting UML in the Design ofMulti-AgentSystems". *ESAW'00 at 14th European Conference on Artificial Intelligence*. August, 2000.
[16] R. Tolksdorf "Models of Coordination". *ESAW'00 at 14th European Conference on Artificial Intelligence*. August, 2000.
[17] G. Cabri, L. Leonardi, F. Zambonelli, "Context-dependency in Internet-agent Coordination". *ESAW'00 at 14th European Conference on Artificial Intelligence*. August, 2000.
[18] C. Baeumer, M. Breugst, S. Choy, T. Magedanz: "Grasshopper - A Universal Agent Platform Based on OMG MASIF and FIPA Standards". http://www.grasshopper.de/download/ghandstandards.pdf

Author Index

Amor, Mercedes 128

Bergenti, Federico 106
Brueckner, Sven 19

Cabri, Giacomo 51
Castelfranchi, Cristiano 1

Demazeau, Yves 93

Fuentes, Lidia 128

Gruer, Pablo 114

Hilaire, Vincent 114

Koch, Christoph 64
Koukam, Abder 114

Leonardi, Letizia 51

Matthews, Robert S. 19
Moro, Gianluca 34
Müller, Jean-Pierre 114

Parunak, H. Van Dyke 19
Petta, Paolo 64
Pinto, Mónica 128
Poggi, Agostino 106

Ricordel, Pierre-Michel 93

Sauter, John 19

Tolksdorf, Robert 78
Troya, José María 128

Viroli, Mirko 34

Zambonelli, Franco 51